Master's Degree in Informatics Engineering
Thesis
Final Report

Data Science for Non-Programmers

Orchestration of Microservices and Graphical User Interface

Bruno Leonel André Lopes
bllopes@student.dei.uc.pt

Supervisors:
Prof. António Jorge Silva Cardoso

Prof. Filipe João Boavida Mendonça Machado de Araújo

Prof. Rui Pedro Pinto de Carvalho e Paiva

September 3, 2018

FCTUC FACULDADE DE CIÊNCIAS
E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

This page is intentionally left blank.

Abstract

With the emergence of Big Data, the scarcity of data scientists to analyse all the data being produced in different domains became evident. To train new data scientists and make experiments with data faster, web applications providing good data science practices without requiring programming skills can be a great help. However, some available web applications lack in providing good data mining practices, specially for the assessment and selection of models. Thus, in this dissertation we describe a system, currently under development, that will allow the construction of data mining workflows enforcing good data mining practices. The main technical challenges addressed in this thesis were the adoption of **Netflix Conductor** to orchestrate the microservices; the development of **Condu**, a Python utility library to interface with Conductor; and the construction of the **graphical user interface**. Usability tests, were conducted with two groups of users to evaluate the envisioned concept for the creation of data mining processes. In these tests we observed a general high level of user satisfaction.

Keywords

Data Science, Data Mining, Microservices, Orchestration, Workflows

This page is intentionally left blank.

Resumo

Com o surgimento de *Big Data*, a escassez de *data scientists* para analisar todos os dados produzidos em diferentes domínios tornou-se evidente. Para treinar novos *data scientists* e para fazer experiências mais rápidamente, aplicações web que providenciem boas práticas de *data science* e que não requeiram experiência em programação, podem ser uma grande ajuda. Contudo, algumas aplicações web não aplicam bem as boas práticas de *data mining*, especialmente na avaliação e seleção de modelos. Assim sendo, nesta dissertação iremos descrever um sistema, atualmente em desenvolvimento, que permitirá a criação de *worflows* com especial ênfase nas boas práticas de *data mining*. Os desafios tecnológicos principais abordados nesta tese, foram a adoção do **Netflix Conductor** para a orquestração de *microservices*; desenvolvimento da biblioteca **Condu**, que facilita a comunicação com o *Conductor* escrita em Python; e a construção de uma **interface gráfica** para o utilizador. Testes de usabilidade foram feitos com dois grupos de utilizadores para avaliar o nosso conceito de criação de processos de **data mining**. Nestes testes foram alcançados altos níveis de satisfação por parte dos utilizadores.

Keywords

Data Science, Data Mining, Microservices, Orchestration, Workflows

This page is intentionally left blank.

Acknowledgements

The accomplishments of this dissertation were only made possible by the important contributions and incentives of a couple of people to whom I am tremendously grateful.

I would like to express my very great appreciation to my supervisors Prof. Rui Paiva, Prof. Filipe Araújo and Prof. António Cardoso, for their guidance and their support during the academic year. I would also like to thank Eng. Jaime Correia for his advice and help in the technical challenges of cloud computation, and Artur Pedroso as my project partner who was part of the development team.

To my friends and colleagues who always stood by my side during this intense and stressful phase, I give a huge thank you. I could not go by without mentioning my family, specially my parents and uncles, for their unconditional support during all my life and for always incentivizing me to go further and work harder.

Last, but not least, I would also like to offer my special thanks to Professors Joel Arrais and Jaime Ramos, and their students, for helping and participating in the usability tests.

This page is intentionally left blank.

Contents

List of Figures

This page is intentionally left blank.

List of Tables

This page is intentionally left blank.

Acronyms

CLI Command Line Interface. 9

CNA Cloud Native Applications. 1

DS4NP Data Science for Non-Programmers. 1, 3, 19, 21, 43, 51, 81

DSaaS Data Science as a Service. 2

FAQ Frequently Asked Questions. 8

FOSS Free and Open Source Software. 44

GUI Grafical User Interface. ix, 2, 8–14, 41, 51, 69, 83

HTTP HyperText Transfer Protocol. 65

JWT Json Web Token. 54

ML Machine Learning. 1, 5–15, 17, 64, 85

NFS Network File System. 80

RAM Random-Access Memory. 22

REST REpresentational State Transfer. 41

RPC Remote Procedure Calls. 41

SaaS Software as a Service. 6

SOA Service Oriented Architecture. 45

SOAP Simple Object Access Protocol. 13, 41

SVM Support Vector Machine. 6, 10–12, 58

This page is intentionally left blank.

Chapter 1

Introduction

This thesis was carried out by the candidate Bruno Lopes, under the project PTDC/EEI-ESS/1189/2014 — Data Science for Non-Programmers, supported by COMPETE 2020, Portugal 2020-POCI, UE-FEDER and FCT. It took place in the Department of Informatics Engineering at the University of Coimbra. In this chapter we present the motivation and scope for the project, followed by the objectives and approaches specific to this thesis. The last section presents the report structure.

1.1 Motivation and scope

In a broad view, data science is the process of discovering interesting patterns and knowledge from large amounts of data. The term 'data science' is commonly used to refer to several techniques that are used for the extraction of information; such techniques include data mining and Machine Learning (ML). Those techniques often intersect with each other and often used interchangeably. ML relates with the system's ability to 'learn' from data; data mining refers to the process from data extraction to its analysis, also including ML [24].

For the correct application of data mining processes and also for the evolution of the field, competent data scientists are required. They are a resource in high demand these days [34]. As the consulting company McKinsey estimates, the main reason can be attributed to the lack of qualified workers to fulfil the roles that are currently open [25]. To fill such demand, more data scientists need to be trained, which requires time, due to the diversity of disciplines to learn [11]. By abstracting somehow programming languages from the data scientists' path, we might reduce the necessary time to train them. Hence, the objective of the Data Science for Non-Programmers (DS4NP) project is to explore the use of visual programming paradigms to enable non-programmers to be part of the data science workforce. In addition, the project aims to take the computation of the workloads to the cloud; as opposed, to the computation performed locally on the users' infrastructure. More specifically, the objective of this project is to build Cloud Native Applications (CNA) for data science using microservices.

1.2 Objectives and Approaches

Having the data mining process in mind, we decided to create a system that allows users to build workflows representing the data mining process. It will be available through a **Grafical User Interface (GUI)** with focus in guiding the user in the creation of workflows without requiring programming skills. The user will be able to create experiments based on workflows composed of sequential data mining tasks. These tasks will allow data insertion, preprocessing, feature selection, model creation and model evaluation. Some tasks will include parameters that can be used in grid search along with nested cross validation, enforcing good model assessment and selection practices [31].

In line with the goals of the project, the system will be built as a cloud application where each set of functionalities will be provided by a single-purposed microservice. The execution of the workflows will be performed by workers that will execute each task independently. The key component for the management of the execution will be the orchestrator, **Netflix Conductor**.

To evaluate the envisioned system, we conducted usability tests using a group of users familiar with data mining, and another group without that experience, though having a background in statistics, whom can also benefit from our software. We observed an overall positive user satisfaction with both groups. To assess the impact of the current microservices architecture in the performance of the system, we deployed it in a public cloud and performed tests using datasets with different sizes. The results are promising and an incentive to guide us in new directions.

1.3 Contributions

The development of this project resulted in the following contributions:

- A microservices architecture for data science using orchestration to manage the execution of workflows.
- The correct implementation of data mining workflows enforcing good practices.
- A GUI that simplifies the creation of workflows.
- A paper entitled 'DataScience4NP - A Data Science Service for Non-Programmers' for the INForum 2018 conference [10].
- A paper entitled 'A Data Mining Service for Non-Programmers' for the KDIR 2018 conference [4].

1.4 Report structure

The remaining document is organised as follows. In chapter 2, we give a detailed account of the state of the art. It characterises platforms that already provide Data Science as a Service (DSaaS), and other tools and research projects that are relevant for data science. After their analysis, we talk about cloud methodologies that will be used in the proposed system and the orchestration engine Conductor.

Chapter 3 contains the specification of the functional requirements through use cases, quality attributes using a utility tree, strategies to achieve the quality attributes, and restrictions imposed on the project.

Chapter 4 characterises the proposed architecture that will be implemented and the technologies used.

Chapter 5 describes the microservices that compose the DS4NP platform in more detail, focusing on the motives behind the technologies adopted and how they work.

Chapter 6 details the tests performed in the system. It is divided into three parts. First, we describe our process for unit testing. Then, the results of the usability tests performed on 18 participants are presented and analysed. The last part describes a performance comparison between our system and H2O.

In chapter 7, we introduce the stakeholders of the project, the development methodology used and the work plan stipulated for the academic year.

Finally, in chapter 8 we draw the main conclusions of this work and point out directions for the future.

This page is intentionally left blank.

Chapter 2

State of the art

There are currently a few data mining systems similar to the one that is implemented and described in the context of this thesis. Previous studies have done critical analysis of such kind of systems. For example, in the article 'Overview of Machine Learning Tools and Libraries' [39], the authors surveyed traditional Machine Learning (ML) products considering a list of properties. They took into account license models, supported operating systems, ability to handle big datasets, language support, classes of machine learning problems supported, usability, among others. In relation to classes of machine learning problems supported, they identified scikit-learn, Weka, R and RapidMiner as the most versatile software packages, supporting tens of ML methods. In relation to usability, Orange and the proprietary solution IBM SPSS Modeler were the ones providing best results. They also identified Knime, RapidMiner, Salford Systems, or SAS Enterprise Miner as being able to execute data analysis tasks in distributed environments. The study concludes by discussing the challenges raised by applying popular ML algorithms to large amounts of data. The main challenge identified was the lack of support in processing big data sets, for which new approaches based on parallelization of time-consuming tasks are needed.

Having in mind such challenges, one of the authors of the previous study, Daniel Pop, in a subsequent study [38] investigated the impact of the Cloud computing paradigm in the field of ML. In this later study he identifies the movement towards the Cloud as being mainly motivated by the increasing size of datasets and the difficulties they impose in traditional ML systems. The study observes that the cloud computing paradigm and cloud providers turned out to be valuable alternatives to speed-up ML platforms. Several new solutions following the cloud computing trend are then presented.

In an attempt to extend and update the previous studies, in the next section we will present our own research related with current data mining systems. In this analysis our main focus is to present the systems, but mostly to identify possible issues that are observed in order to address them in our implementation.

2.1 Related Frameworks

In the following subsections we give a brief overview of data science frameworks that also provide data science functionalities.

2.1.1 AzureML

AzureML [1], is a Software as a Service (SaaS) solution that enables collaborative construction of knowledge discovery solutions using a drag and drop interface. It lets users perform different ML steps from data preparation to model deployment.

The different ML steps can be applied by constructing workflows composed by different building blocks. In these blocks the user can upload data or connect to data already in the cloud. He can then continue to use building blocks to perform different ML tasks, using the previously inserted data. These ML tasks can include preprocessing, feature selection, model creation and model evaluation, which are employed using several algorithms provided by Microsoft. Finally, a created model can be published as a web service and becomes accessible for future use.

In Figure 2.1, an example of a workflow created in AzureML is illustrated, containing cross-validation to create an Support Vector Machine (SVM) model.

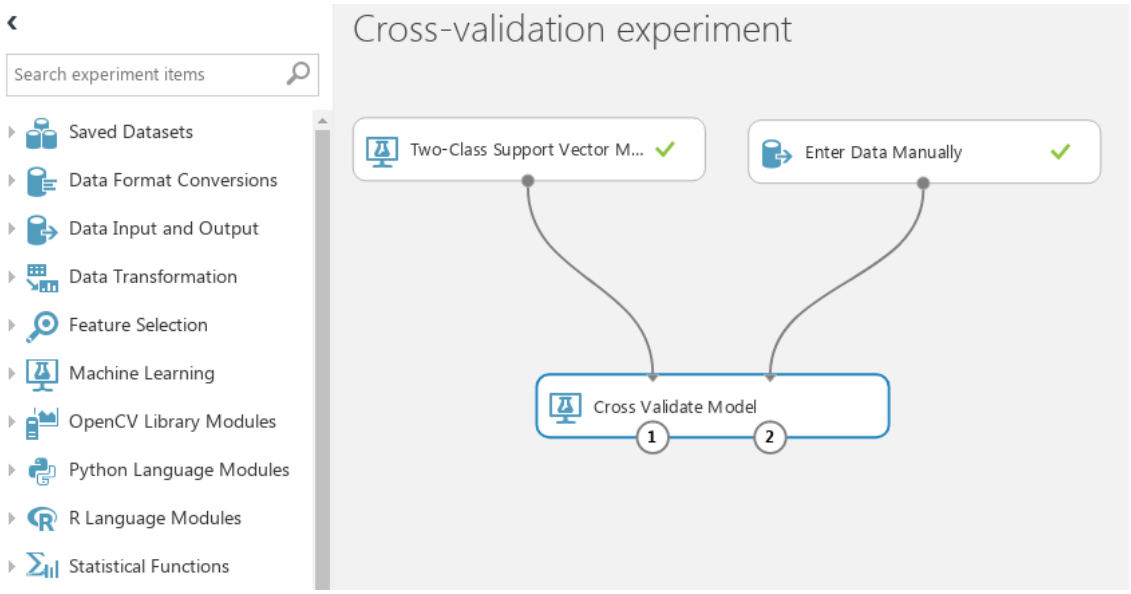

Figure 2.1: AzureML workflow example.

Azure ML does not require any programming skills and provides data science templates for specific domains, such as maintenance prediction, retail customer churn prediction, online fraud detection, retail forecasting, and text classification.

Azure ML provides a few well-known filter feature selection methods, including Pearson correlation and mutual information, among others. Besides these well-known filter methods it also provides a custom one, which they call Permutation Feature Importance. Fisher Linear Discriminant Analysis is another feature selection method provided by AzureML, however this method transforms the features to a new space.

[1] https://studio.azureml.net/

Azure ML, does not offer any option to perform automated feature selection, using wrapper methods, though they have high relevance in ML.

AzureML requires the users to build complex workflows, with multiple connections between blocks, to create models with different features and parameters. Moreover, it is not possible to apply cross-validation correctly in workflows that involve preprocessing or filter selection prior to train/test the final model. Basically, in this platform the preprocessing and feature selection phases must be applied to the entire dataset in use, prior to training/testing the final model using the cross-validation procedure, which is a wrong practice [12, 22, 31] and may provide overly-optimistic error estimates for the produced models.

Besides not employing cross-validation correctly for all cases, they also do not support nested cross-validation and the solution is proprietary.

2.1.2 AmazonML

AmazonML [2] takes a different approach to providing machine learning as a service. Instead of providing many different tasks to create ML models, it is focused on classification and regression problems only. In Amazon's perspective, these problems should not require the user to specify what ML tasks should be performed to train/test a model. The steps are very straightforward: the user uploads a dataset in a supported format (usually csv), selects the attribute that he wants to predict (the class) and Amazon takes care of the rest.

In Figure 2.1, the steps required to create a model from a dataset and make predictions are illustrated.

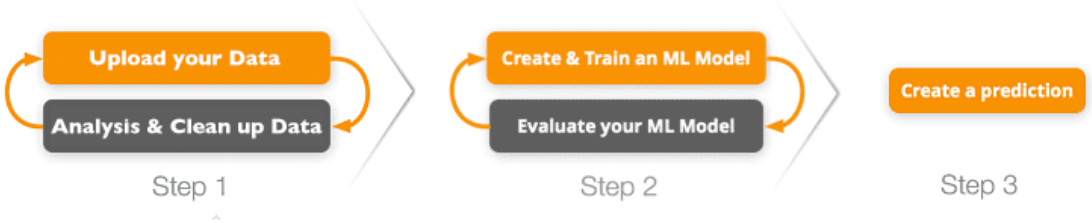

Figure 2.2: Steps to build a model in AmazonML

This solution allows the user to create models and predictions easily and with a low amount of effort. However this simplified solution brings many downsides, as it does not allow the user to configure any ML task to be applied to its data. The user needs to trust that the AmazonML platform will really use the best techniques to provide a final model. Moreover, this is a proprietary solution.

[2] https://aws.amazon.com/aml/

2.1.3 H2O

H2O [3] emerged in 2011 with the goal of democratising data science by making scalable ML open source and accessible to everyone. It is now in version 3 and offers a fully distributed in-memory ML open source platform with linear scalability. The platform can be used from a web UI that gives the possibility to apply ML in a sequence of steps without requiring users to write any code. It can also be used from programming environments like R, Python, Java. H2O also developed Sparkling Water that enables H20 to be used in the Spark platform. H2O provides implementations of many popular algorithms such as Gradient Boosting Machines, Random Forest, Deep Neural Networks, Word2Vec, Stacked Ensembles, Naive Bayes and Linear Models. H2O is extensible, enabling developers to add data transformations and custom algorithms of their choice and access them through the clients.

To make ML more user-friendly and easy for non-experts, H20 provides AutoML. AutoML is a feature that automates the ML workflow by automating training and tuning of many models within a user-specific time-limit or number of models that can be selected. H2O is a good solution to build ML models using Big Data and to put them into production.

In Figure 2.3, an example of how to create a Naive Bayes model in H2O is presented.

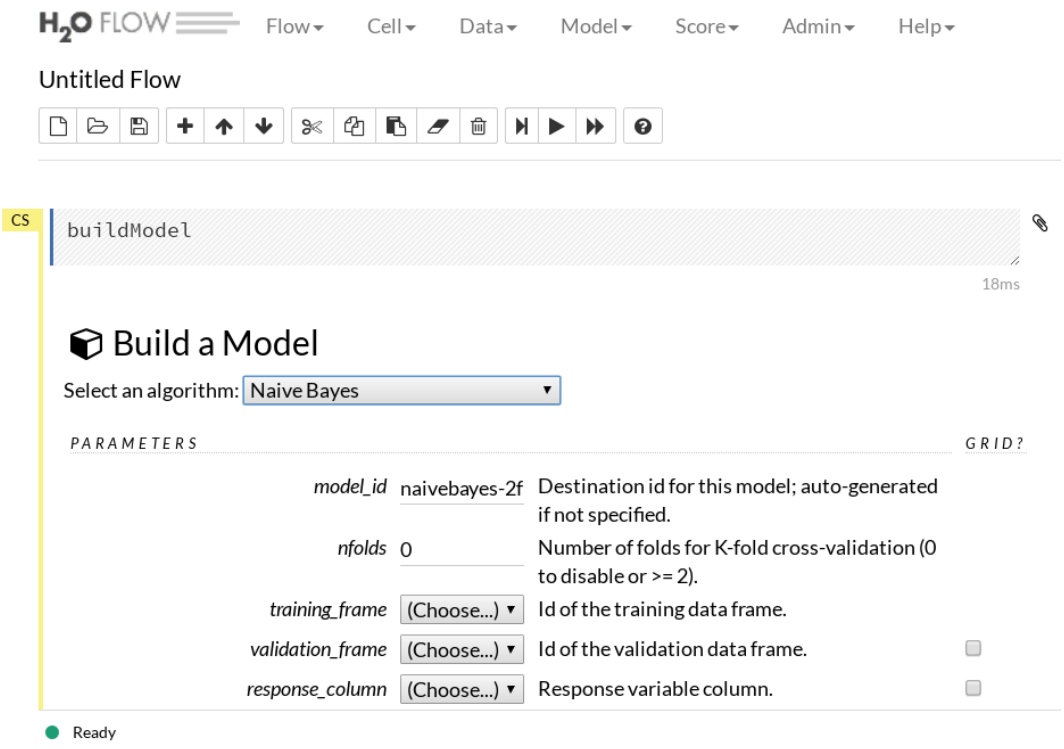

Figure 2.3: Example of the H2O Grafical User Interface (GUI)

On the down side, H2O does not provide High Availability. Whenever a node in the cluster becomes unavailable, it requires the whole cluster to be rebooted, to include the node again. According to H2O Frequently Asked Questions (FAQ): "If a node in the cluster is unavailable, bring the cluster down and create a new healthy cluster" [20]. Thus, adding more nodes or changing the current functionalities of H2O requires a reboot of the cluster. According to other FAQ: "New nodes can only be added if H2O has not started any jobs.

[3] https://www.h2o.ai/products/h2o/

Once H2O starts a task, it locks the cluster to prevent new nodes from joining. If H2O has started a job, you must create a new cluster to include additional nodes". Thus, if the number of ML workflows to execute in the system increases there is no way to scale the platform without rebooting the cluster. Platform rebooting is also required when it is necessary to insert new functionalities in the platform, such as new ML algorithms.

Besides H2O allowing users to train/test models in the Flow UI without requiring programming skills, there is no option to perform features selection or data preprocessing prior to the model creation.

The prototype depicted in this thesis, will both enable users to perform preprocessing and feature selection tasks prior to model creation and will also enable the insertion/deletion of services and nodes in runtime, without requiring the whole cluster to be rebooted.

2.1.4 Google Cloud Machine Learning Engine

Google Cloud Machine Learning Engine [4] runs in Google Cloud Platform and lets the user build machine learning models that work with data of any type and size, using the TensorFlow framework [5]. They offer large scale training of models in managed clusters. After training a model, they can be used for batch and online predictions also in a managed cluster. All models become immediately available in their global prediction platform, which supports thousands of users and TBs of data.

Google Cloud Machine Learning system is in fact a very strong system to run ML works, however it does not provide a GUI. It requires the users to create a script with all the steps necessary to train a model using one of the supported frameworks (e.g., Tensorflow). Then, the users need to send the script for execution in the cloud environment using the Google Cloud Command Line Interface (CLI). Moreover, this is proprietary solution.

[4] https://cloud.google.com/ml-engine/
[5] https://www.tensorflow.org/

2.1.5 Weka

Weka [6] is a very popular software among data scientists, used to apply a great variety of data mining and ML tasks through a GUI. Alternatively, users can also apply algorithms by writing programs in Java, using Weka's extensive library. The provided algorithms include various implementations for regression, classification, clustering, association, rule mining and attribute selection. The software also provides many data preprocessing and visualisation functionalities.

The user can perform ML tasks individually, or in sequence by building workflows with desired tasks. To build a workflow describing the whole ML process, the user creates a sequence of tasks in the GUI by dragging available ML blocks to the workspace. Then, the different blocks can be connected in the desired order.

In Figure 2.4, we can see an example of a workflow to create an SVM model, assessed using cross-validation.

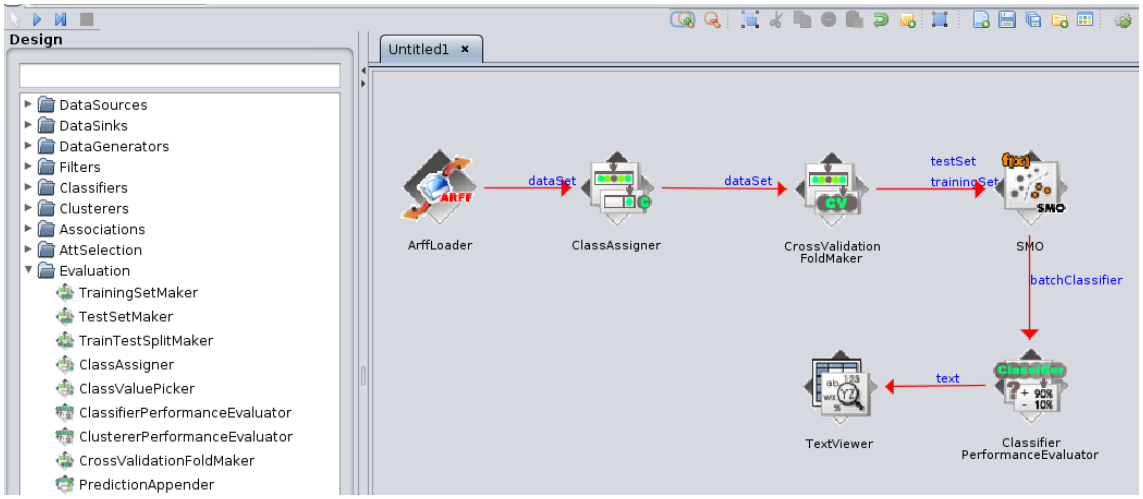

Figure 2.4: Example of a workflow in Weka

Although the software allows the users to create ML processes without requiring programming skills, the number of tasks that is necessary in the workspace can be very large and it is not explicit for a novice user what different tasks and connections must be created. Moreover, this software requires being installed in the user's computer. Thus, it does not take advantage of cloud properties and is not primarily intended to be used with Big Data, though it seems to support it [41].

[6] https://www.cs.waikato.ac.nz/ml/weka/

2.1.6 Orange

Orange [7] is a software that allows users to create iterative ML workflows using a visual programming paradigm, in which users drag blocks representing specific ML tasks to a working area and connect these blocks to compose a workflow.

This software is characterised by delivering great data visualisations to their users and providing other functionalities, such as data preprocessing, model creation and model evaluation.

Overall, its functionalities are similar to the ones provided by Weka and AzureML, though with a more pleasant and easier to use GUI.

In Figure 2.5, an example of an SVM model trained and tested using the cross-validation procedure is presented.

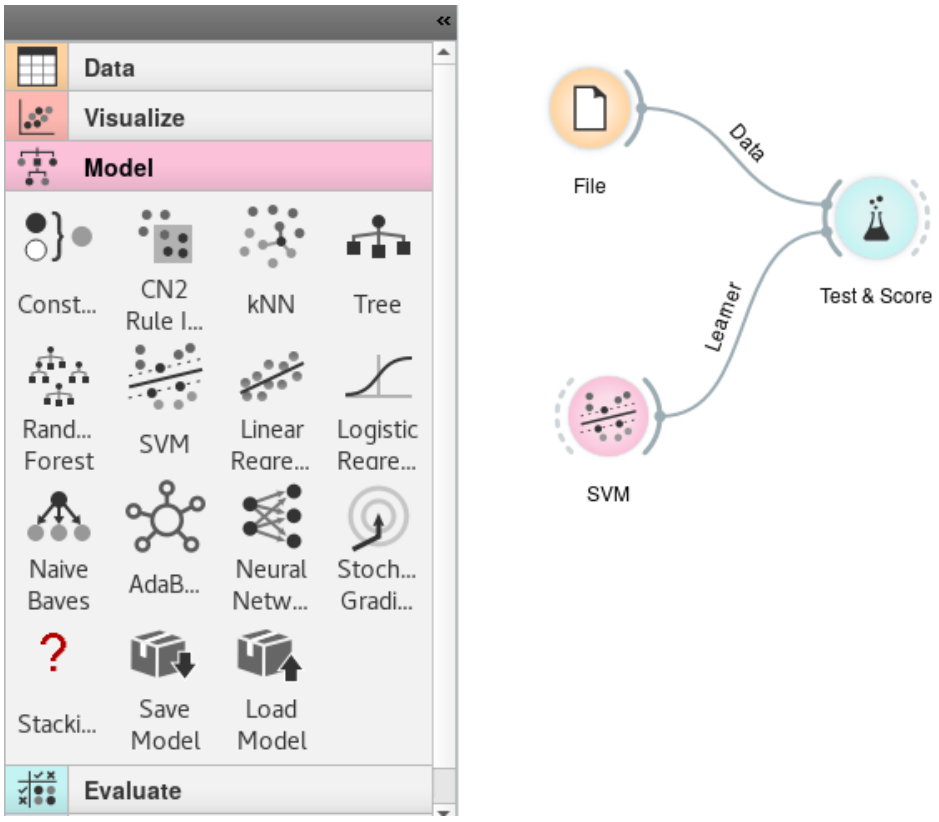

Figure 2.5: Example of a workflow in Orange

Although this software is very intuitive and focused in solving a problem similar to ours, it is a standalone system, requiring users to install it in their machines. Similarly to AzureMl and Weka, this application also requires the user to insert several building blocks with multiple connections to conduct the ML experiments. Moreover, this solution does not offer nested cross-validation to train/test ML models.

[7] https://orange.biolab.si/

2.1.7 RapidMiner

Rapidminer [8] follows a concept similar to Weka, Orange or AzureML. It allows users to create ML processes by dragging different building blocks, representing ML tasks, into a workspace, where these blocks must be connected in a desired way.

Concerned about wrong ways to assess and select models, seen in other applications, such as AzureML, RapidMiner offers methods like nested cross-validation for model assessment and selection.

Similarly to Weka and Orange, to use this tool it is also necessary to install it locally. In addition, RapidMiner also allows the ML workflows to execute on a cluster, using the installed GUI. It presents the same challenges as AzureML, Weka or Orange to create ML workflows, as it uses the concept of dragging blocks to the workspace and making connections between these blocks.

In Figure 2.6, an example of an SVM model trained and tested using the cross-validation procedure is presented.

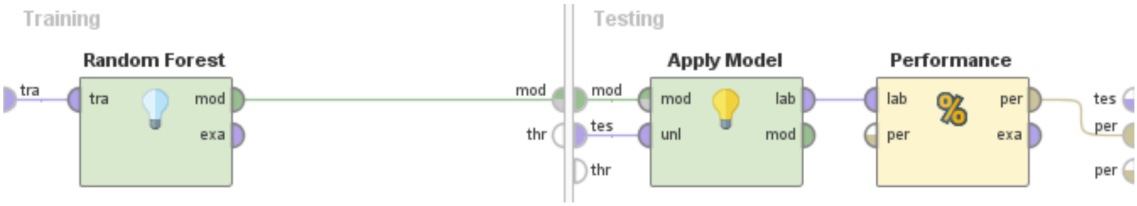

Figure 2.6: Example of a workflow in Rapidminer

RapidMiner can be used for free, though with many limitations. In the free version it restricts the use of datasets with a maximum number of 10000 instances, and the computations can only use a single processor. To use RapidMiner without limits requires payments in the order of tenths of thousands of dollars a year per user.

ClowdFlows

ClowdFlows started as a research project [30] and is currently available online at clowdflows.org. It is a system that enables non-programmers to build data mining workflows using a visual programming paradigm and to share workflows online.

The system can process Big Data in batch or real-time mode through an ML library provided by the researcher on top of the Disco[9] MapReduce framework. Besides providing the ML library for Big Data, they also integrate major ML libraries like Weka, Orange and scikit-learn to be used from the GUI. The execution of the workflows is done through worker nodes that will get tasks to execute using a message queue; this was implemented using the message broker RabbitMQ [37].

Besides the algorithms that are already present, ClowdFlows also facilitates the introduction of new algorithms. The users just need to create a web service with an interface that follows the system specification. Next, using the GUI is possible to point to the URL of the WSDL that describes the Web Service and it becomes available to be used in a

[8] https://rapidminer.com/
[9] http://discoproject.org/

workflow. This extension can also be done by creating Python packages but requires the user to have ClowdFlows installed locally.

In Figure 2.7, an example of a workflow created in ClowdFlows is presented.

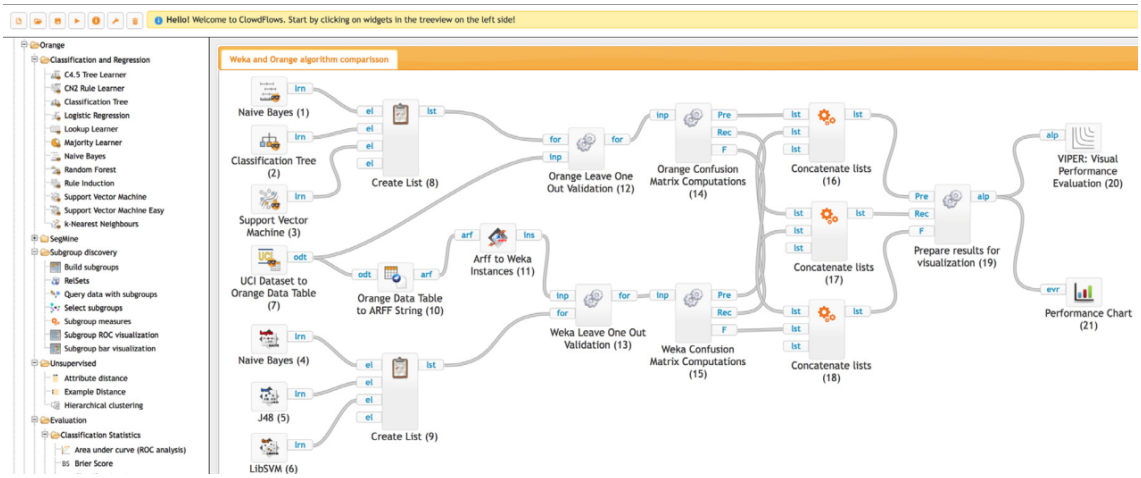

Figure 2.7: Representation of a workflow in the ClowdFlows platform.

This system presents the same drawbacks seen in other platforms, such as AzureML, Weka and Orange. To train/test models, it requires the users to build complex workflows composed by different building blocks and connections, with functionalities that are not always clear. Similarly to other platforms, the cross-validation process included in this platform is only applied to train/test the final model. Nested cross-validation is also not included in this platform.

DAMIS

DAMIS, another research project [33], is a cloud solution for data mining. It provides access through the web site damis.lt and gives the user the possibility to build graphical workflows for knowledge discovery using a drag and drop interface, similarly to previous plaforms, such as AzureML, Orange or ClowdFlows. The ML tasks used in the workflows built in the GUI are sent to a computational service using Simple Object Access Protocol (SOAP) messages, which after validating the request pushes it to a job scheduler of a specific computational infrastructures.

The ML tasks provided in this platform are specially focused in dimensionality reduction. However, they also employ other algorithms for data preprocessing, classification and clustering, such as data cleaning, feature selection, normalisation, splitting, outliers filtering, computation of data statistics, multi-layer perceptron, random decision forest, k-means, and self-organising maps.

Figure 2.8 shows an example of a workflow created in DAMIS. This example was extracted from the DAMIS paper, as it was not possible to enter in the DAMIS workspace during this thesis.

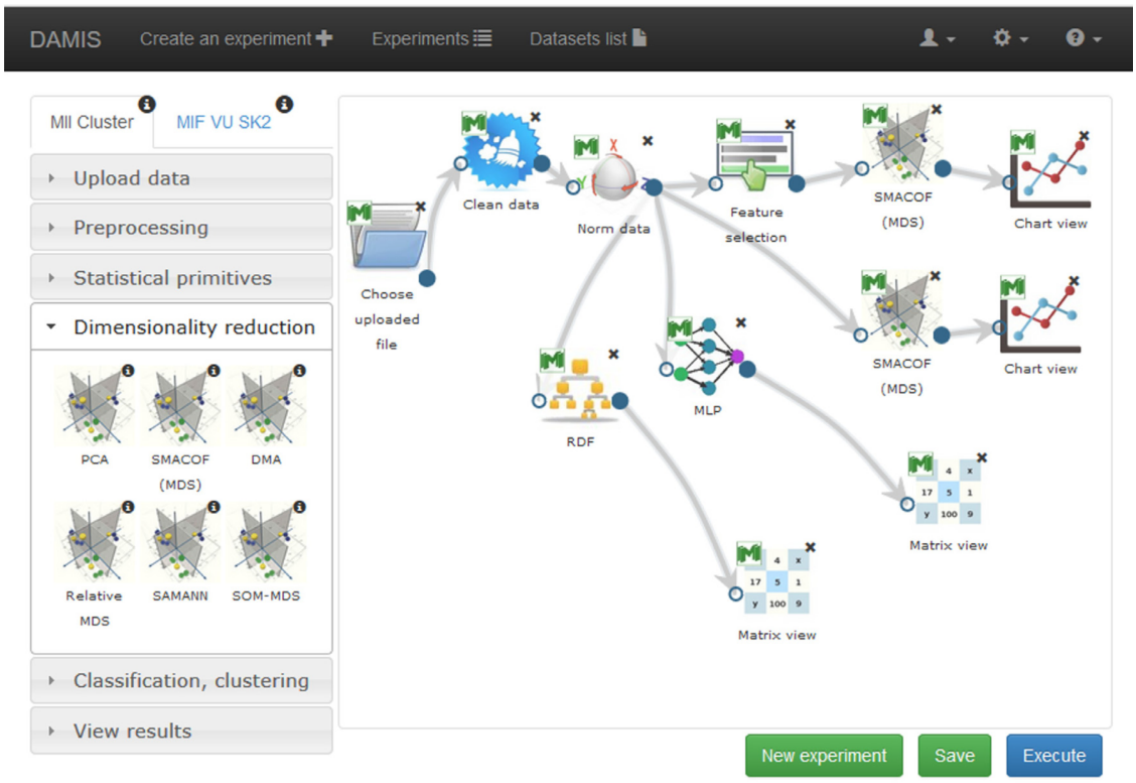

Figure 2.8: Representation of a workflow in the DAMIS platform.

After reading the DAMIS paper, this platform does not seem to employ great ML practices to train/test ML models, as they never refer how the trained models are tested.

2.1.8 Comparison of Data Mining and Machine Learning solutions

To evaluate the differences between the data mining and ML solutions presented before, here is included a table with the previous platforms along with ten attributes:

- **Open Source**: signals if the solution is open source;
- **Cloud**: signals if the solution is hosted in the Cloud or has some functionalities that are computed in a Cloud environment;
- **GUI**: signals if the solution has a GUI;
- **Flexibility**: while some solutions provide a large number of different ML tasks to create ML processes, other solutions are more straightforward and do not give much flexibility to the user. Here, it is used a scale from 1 to 5 to classify the flexibility of each solution, where number 1 is the minimum flexibility and number 5 the maximum flexibility;
- **Usability**: by analysing each GUI we classified, in a scale from 1 to 5, the usability of each solution according to the following aspects: attractiveness, intuitiveness, complexity, efficiency.
- **Big Data**: signals if the solution is capable of processing large amounts of data;
- **High Availability**: signals if the solution was designed to support High Availability;

- **Extensibility**: signals if the solution allows the addition of new ML functionalities.
- **Templates**: signals if the solution has predefined templates to execute ML processes;
- **Correct CV**: signals if the solution includes good practices to train/test models using cross-validation and nested cross-validation, when preprocessing tasks (e.g., normalisation, feature selection) are also included in the process.

The attributes that have question marks ('?') mean that we were unable to determine the attribute in question. The attributes with the N/A value are attributes that are not applicable to the solution in question.

System	Open Source	Cloud	GUI	Flexibility	Usability	Big Data	High Availability	Extensibility	Templates	Correct CV
AzureML		x	x	4	3	x	x	x	x	
AmazonML		x	x	1	5	x	x		N/A	?
H2O	x	x	x	3	3	x		x		
Google Cloud ML Engine		x		N/A	1	x	x	x		N/A
Weka	x		x	4	3	x	N/A	x		x
Orange	x		x	3	4	?	N/A	x		x
RapidMiner		x	x	5	3	x	x	?	x	x
ClowdFlows	x	x	x	4	2	x	?	x	x	?
DAMIS	x	x	x	2	?	x	?	x		?

Table 2.1: Comparison of Data Mining and Machine Learning solutions

As seen in Table 2.1, and from the analysis that was previously done in the sections related to each solution, we can observe that, with the exception of Weka, Orange and RapidMiner, no other solution enables the best practices for cross-validation when using preprocessing or feature selection tasks prior to the train/test of the final model. From these solutions, only Weka and RapidMiner provide nested cross-validation. The Google Cloud ML engine does not apply (N/A) to this attribute because, the ML process needs to be programmed, making the correct practice of cross-validation a responsibility of the programmer.

Related to the usability of each solution and their flexibility, we observed an interesting result. Each solution, in order to provide more flexibility to the user, sacrifices its usability. AmazonML had the highest usability given the fact that the user specifies a dataset and the rest of the ML process is done automatically; however, the user can not specify what algorithms should be used, or if the process should include tasks for preprocessing or feature selection prior to training the final model.

In contrast, RapidMiner offers high flexibility as it has a big number of ML tasks that can be inserted in the workflows with multiple configurations, however decreasing the usability, as these tasks need to be properly connected to create a proper ML experiment.

The above mentioned attributes are the main topics approached in our solution. We will provide the correct application of cross-validation processes including preprocessing and feature selection tasks prior to train/test a model. Nested cross-validation will also be addressed and we will also introduce the possibility of specifying the number of repetitions to do both cross-validation and nested cross-validation, which is not provided even in RapidMiner. We will address the usability and flexibility aspect by providing the construction of simple sequential ML workflows, however allowing the user to have much flexibility as s/he can select very different tasks of the preprocessing, feature selection, learning and evaluation phases.

In relation to the other attributes, we can observe that each solution has different characteristics. Some solutions are cloud-based, others open-source, few have templates. In the envisioned prototype we will address all of these attributes.

From our analysis RapidMiner appears to be the best solution in relation to the attributes presented in the table, thus it is our most direct concurrent. However, as it was also mentioned before, this solution is expensive.

Weka can also compete with our solution. However it does not run on the cloud and has lower usability.

Clowdflows addresses much of the previously mentioned attributes. However, the construction of workflows in this tool is extremely complex and does not provide much information. We were not even aware if it can correctly train/test models employing correct cross-validation or nested cross-validation.

Orange is an other solution that can compete with ours, though, similarly to Weka, it does not run in the cloud. Thus, the properties provided by the cloud, such as the execution of ML experiments remotely, are not present. Moreover, in this solution it is not possible to use nested cross-validation.

2.2 Cloud Computing

This section describes the methods and technologies researched to accomplish our objectives of cloud computation. Starting with the advantages of microservices, followed by containerisation and finishing with orchestration.

2.2.1 Microservices

With the purpose of providing many different algorithms to the users for their data science projects, one wouldn't want to create a monolithic architecture. This approach would increase the difficulty to maintain the system, reduce the agility of the system to adapt to newer business needs, such as new algorithms and features, and make it less scalable. Contrarily to a monolith, microservices are a paradigm in which a monolithic application is decomposed into small micro applications. This type of architecture is emerging as an important approach for distributed mission-critical applications. In a microservice-based architecture, the application is built on a collection of services that can be developed, tested, deployed, and versioned independently [13].

Advantages of a microservices architecture:

- As the services are small and decoupled, it is easier to write and modify them, lowering the **maintenance** cost.
- When failures happen, they are constrained to that microservice and affect only its functionalities. In some cases, that microservice might get shutdown and a new fresh one take its place. Keeping the work and failures contained helps insuring **reliability** and **modularity**.
- **Scaling** is supported by creating more replicas of microservices.
- The deployment of microservices is also easier because of their loose coupling with the rest of the system. Moreover, due to the lightweight property of microservices, deployment is also fast and changes in code are less error prone.
- Due to their lightweight properties and fast replacement, the system downtime will decrease and the platform becomes highly **available**.

Even tough most of the microservices were developed using the Python language, due to its simplicity and popular use in data science, a microservices architecture is fundamentally language agnostic. We mean by this is that since each service communicates using a well-know interface that is accessible regardless of the programming language, each service could be implemented using a different languages if it brought more value. In our case, using the same language for most of the microservices allowed us to share some portions of the code that was similar in different services.

Most of the microservices that were implemented needed a database to store their resources. One of the challenges we faced was the unstructured nature of the data mining workflows. They have many tasks with different fields. Each task can have a completely different structure from each other. For this reason we adopted a database for unstructured data. After an analysis of the most relevant databases to overcome this challenge we found MongoDB [10] as the most agile in terms of allowing the schema of the data to

[10]https://www.mongodb.com/

change quickly as we develop each microservice. Since the microservices are known to change quickly we needed a database that could reflect this. Hence, our adoption of this database for most of the microservices.

2.2.2 Containerisation

In addition to the use of microservices, enterprises are increasingly performing cost savings, solving deployment problems, improving DevOps and production operations by containerising microservices [13]. A container is a lightweight, stand-alone, executable package of a piece of software that includes everything needed to run it: code, runtime, system tools, system libraries, settings. Puting the microservices in containers will help achieve their deployability, isolation and guarantee their modularity. In this thesis, the technology adopted for containerisation is **Docker** [17]. Its format and container runtime quickly emerged as the *de facto* standard following its release as an open source project in 2013 [14], making the case for its adoption in this project. We studied other alternatives such as **CoreOS rkt** and **LXC**. However, containerisation is much easier in Docker and they have a huge hub of already containerised applications in contrast with all the other technologies.

For load balancing, replication, scaling, elasticity, availability and the other attributes related with managing the infrastructure, we identified two promising candidates, **Docker Swarm** and **Kubernetes**. Both can ensure that we satisfy all the attributes. Docker Swarm is more simple to configure than Kubernetes. However, what it gains in simplicity it lacks in more functionalities. The last allow us to do a more advanced tuning for each microservice (container), and therefore we are able to increase performance and waste less computing resources.

2.2.3 Orchestration

In this section we will explain the need for orchestration and describe the key technologies that allowed us to orchestrate the microservices in the Data Science for Non-Programmers (DS4NP) platform.

Orchestration vs Choreography

Each microservice only performs a specific task and we need to coordinate those tasks to achieve our goal. The specification of the tasks, their behaviour and coordination is called a **workflow**.

There are two ways to achieve this: we could introduce logic into the microservices and make them aware of the whole workflow (**choreography**) or have a centralised way of doing this (**orchestration**).

Choreography means that each microservice would be responsible for calling the next one after the successful completion of its task. Using choreography increases the complexity of the microservices by having the workflow embedded in the code, making them not so micro. It makes harder the task of changing the code of each microservice and there is no way to systematically answer "How much are we done with workflow X?".

A better solution would be a centralised way of coordinating the services, hence the adoption of an **orchestrator**. With this approach, the orchestrator would be responsible with the execution of the workflows and the microservices would only worry about doing their

tasks. The orchestrator would provide visibility and traceability into the state of workflows and allow greater reuse of existing microservices providing an easier path for integrating new microservices.

Conductor

Netflix provides streaming of media and video-on-demand online with more than 120 millions of subscribers world wide [8]. Given the fact that some of their services also follow a microservices architecture, the content platform engineering team produced and open-sourced **Conductor** to orchestrate their workflows [5]. Conductor is used internally to serve millions of requests every day, making it battle tested and reliable.

Here is how Conductor works:

Conductor is an orchestration engine that allows us to create complex workflows in which individual tasks are implemented by a microservice (workers).

Workflows are defined in Conductor using **json** objects. Every workflow is decomposed in **tasks**, where each task is a specific goal that needs to be accomplished to finish the workflow. The tasks are either control tasks (fork or if statements) or application tasks (e.g. feature scaling) that are executed by the **workers**. Worker tasks are implemented by a microservice that runs in a separate environment from Conductor. The workers talk to the Conductor server via REST api endpoints to poll for tasks and update their status after execution. Workers can be written in any language as long as they can poll and update the task results via HTTP endpoints.

Figure 2.9 details their possible states and transitions.

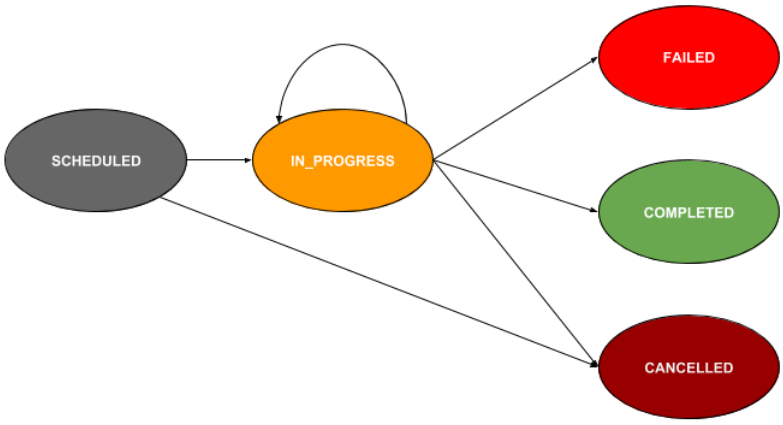

Figure 2.9: Tasks state graph

Conductor servers are stateless and its scalability and high availability can be achieved by replicating Conductor and Dynomite.

Dynomite allows the implementation of high availability and cross-datacenter replication on storage engines that do not inherently provide that functionality. The storage engine used for Dynomite was the in-memory database **Redis**. Its purpose in Conductor is to keep track of the state of all the tasks that are running and the definitions of workflows (blueprints) and tasks.

Figure 2.10 provides an overview of the components that make conductor:

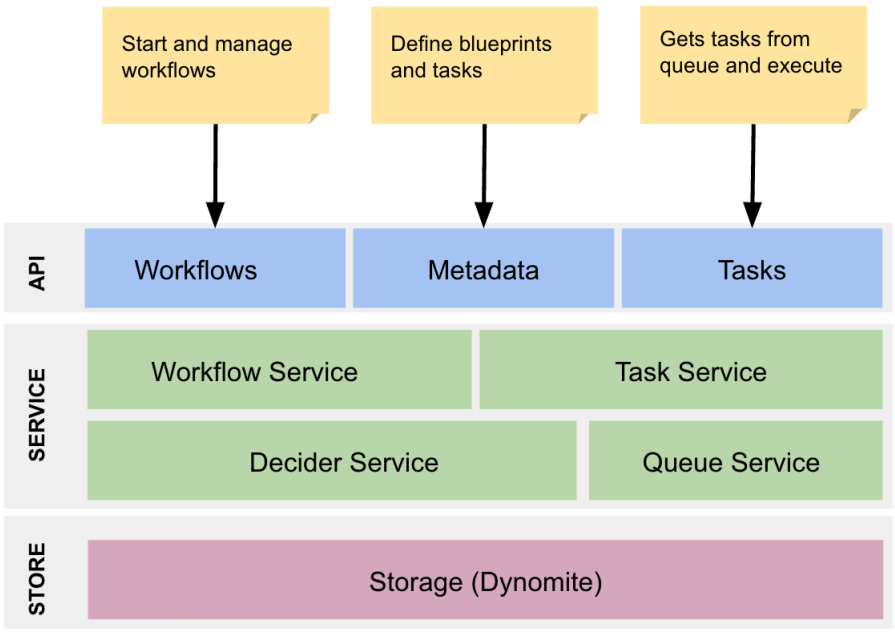

Figure 2.10: Conductor Architecture Overview

The **Clowdflows** platform described in section 2.1.7 also executes its workflows in a similar manner. Both use an in-memory database and follow the producer / consumer pattern to execute workflows. The main difference between the two is that Clowdflows developed their orchestration solution internally and used the message queue broker **RabbitMQ** [37] to send the tasks to the workers. Conductor does this transparently. A message broker such as RabbitMQ could be used instead of Conductor. Howbeit, like Clowdflows, we would still need to implement the orchestration functionalities ourselves. These functionalities include allowing the creation of workflows, scheduling tasks and implementing mechanisms to tolerate failures. In the end, to go with this solution, would be basically implementing the orchestrator ourselves, hence choosing Conductor was a better choice for the prototype. We were not able to find any other solution that offered similar functionalities as Conductor. However, since the DS4NP platform is very modular, changing the orchestration solution for a better one would not need significant architectural changes.

Despite the fact that Conductor works very well in the system, there are some drawbacks that need to be addressed in the future to allow the DS4NP platform to be production ready. The next steps to make that happen should be to address the following drawbacks:

- Due to the nature of workflows and its complexity, we are not able to determine the time it is required for a task to be finished with great accuracy; it depends on many factors such as the parameters and the size of datasets used. This prevents us from defining a timeout for the each task that is appropriate and we end up overestimating every time to accommodate that. The timeout is necessary to have a way of rescheduling a task when the microservice that is currently working on that task crashes or is unable to communicate with the orchestrator. To exemplify, if we define a timeout of 30 minutes and the microservice crashes after 2 minutes, the orchestrator will reschedule the task after 28 minutes of crashing. We can observe that this solution wastes time. A solution would be implementing complementary

components that can detect this events and reports them to the orchestrator.

- Every time a workflow is executed, information about its execution such as logs, number of failures, parameters used and the results of the tasks are stored in the in-memory database. This creates a Random-Access Memory (RAM) problem, because the size of the database increases rapidly and RAM is not as cheap as the disk memory that is commonly used in relational databases. The solution we hypothesise for this, is to delete information of older workflows that were executed and no longer needed.

Chapter 3

Requirements

3.1 Requirements Elicitation

After exploring the tasks conducted by data scientists, mainly recurring to the literature and online forums dedicated to this art, it was observed that most of their works can be conducted using similar workflows. These workflows can be executed using programming languages, or alternatively through applications that provide the mechanisms for non-programmers to build such data science workflows. These applications are essential to train new data scientists without requiring them to acquire profound computer science skills. Although these applications reduce the programming effort from data scientists, they impose limits to the users and might incur data scientists to perform machine learning in a wrong way. Applications providing the correct application of machine learning tend to force the users to build complex workflows with multiple connections between the tasks and do not seem to guide the user during the machine learning process. Having in mind the necessities of non-programmers data scientists and some problems associated with current applications, the prototype that is presented in this thesis aims to enable non programmers to build data science workflows, while guiding them in the process. Being data science a wide scope, the current prototype is restricted to providing workflows for the correct application of machine learning, specifically for the creation of classification models. Generally, the machine learning process is composed by preprocessing, learning and evaluation steps.

These steps might include several tasks with dependencies between each other, creating the necessity to be executed in workflows instead of being executed separately. First, data is provided to the tasks that compose the workflow. Data preprocessing tasks help in preparing the data to the subsequent learning phase, in which different machine learning algorithms might be applied using different parameters that produce different models. These models are then evaluated to verify their performances in order to choose the configuration that provides the best results. The preprocessing phase might include several tasks to transform the data given to the workflow. These tasks include normalisation, feature selection and dimensionality reduction. The learning phase employs different machine learning algorithms using different parameters to create several models that are sent to evaluation. The evaluation phase might require the construction of the previous steps using different validation procedures (e.g., cross-validation, hold out, train-validation-test) to obtain correctly assessed and selected models.

The needs presented by non-programmers in applying machine learning using correct practices along with the limitations detected in other applications were the main driver while

eliciting the requirements that will be presented next.

3.2 Functional Requirements

The functional requirements presented in this section were collected continuously during the prototype development and prioritised according to the MoSCoW method. According to this technique, the requirements can be prioritised into the following groups [32]:

- **M** - Must have: requirements that must be satisfied in the final solution. These requirements are non-negotiable and the project will fail without them.
- **S** - Should have: represent a high-priority feature that is not critical to launch, though it is considered to be important and of a high value to users. Such requirements occupy the second place in the priority list.
- **C** - Could have: a requirement that is desirable but not necessary. Could have requirements will be removed first from scope if the project's timescales are at risk.
- **W** - Won't have: a requirement that will not be implemented in a current release but may be included in a future stage of development. Such requirements usually do not affect the project success.

The requirements were represented as casual use cases following the recommendations from Cockburn [15] and Tenenberg [28]. Casual use cases allow us to have a good understanding about the requirements for the application without requiring much effort in writing them, consequently providing more time for other activities such as design and implementation. Casual use cases are recommended for small projects, with four to six people building a system whose worst damage is the loss of a bit of comfort, easily remedied with a phone call [15]. Each use case starts with the words "Use case" and an identifier for the use case. The identifier is followed by a title that describes the goal of the use case.

The use cases presented here have system scope (black-box), meaning that they discuss the software that is to be built. The level of the use cases presented here are at sea level, as they represent user goals. The title ends with a priority according to the MoSCoW method, that is given inside parenthesis. After the title there is a text describing the interactions between the user and the system in order to accomplish the goal described in the title of the use case. The text might also include a final section with things that might go wrong during the interactions between the user and the system.

Use case 1.1 Insert a dataset in the workflow (M)
An authenticated user wants to insert a dataset on the workflow in order to use it in the subsequent tasks. The user has previously uploaded the dataset to the system. The user selects a task to insert the dataset to be used in the workflow. The system provides a task where the user can specify the dataset to be used. The user specifies the dataset to be used. The system shows the attributes that are present in the dataset. The user selects the attributes s/he wants to remove from the dataset and defines the class label of the dataset. The dataset becomes available to be used in the remaining tasks that compose the workflow.
The things that could go wrong are:

- The dataset specified by the user is invalid, in which case the system will not show the attributes present in the dataset.

Use case 1.1.1 Visualise the dataset inserted in a workflow (M)
An authenticated user wants to visualise the dataset used in an executed workflow. The workflow includes a task to insert a dataset and it has executed successfully. The user sends a request to visualise the task outputs. The system outputs the dataset used in the workflow. The user can visualise the dataset.
The things that could go wrong are:

- The task to insert a dataset did not execute successfully. The user sends a request to visualise the task outputs. The system outputs a message with the reason for the task failure.

Use case 1.2 Insert a data preprocessing task on the workflow (M)
An authenticated user wants to insert a data preprocessing task in the workflow in order to apply preprocessing transformations to the dataset used in the workflow. The workflow is already initiated. The user sends a request to insert a data preprocessing task on the workflow. The system provides multiple data preprocessing tasks so that the user can choose. The user chooses one of the available data preprocessing tasks that is then inserted in the workflow in the position selected by the user.

Use case 1.2.1 Insert a (data preprocessing) min-max scaling task on the workflow (M)
An authenticated user wants to insert a min-max scaling task on the workflow in order to scale the data used in the workflow. The workflow is already initiated. The user sends a request to insert a data preprocessing task on the workflow. The system provides multiple data preprocessing tasks so that the user can choose. The user chooses a min-max scaling task. The task is inserted in the workflow in the position selected by the user. The user can input the min and max values to apply the scaling on the dataset.

Use case 1.2.2 Visualise the output of a (data preprocessing) min-max scaling task (M)
An authenticated user wants to visualise the output from a min-max scaling task after it has been executed in a workflow. The workflow includes a min-max scaling task and it has executed successfully. The user sends a request to visualise the task outputs. The system outputs the datasets where the operation was applied. The user can visualise the scaled datasets, the scaled attributes and the scaler object.
The things that could go wrong are:

- The min-max scaling task did not execute successfully. The user sends a request to visualise the task outputs. The system outputs a message with the reason for the task failure.

Use case 1.2.3 Insert a (data preprocessing) z-score normalisation task on the workflow (M)
An authenticated user wants to insert a z-score normalisation task on the workflow in order to normalise the data used in the workflow. The workflow is already initiated. The user sends a request to insert a data preprocessing task on the workflow. The system provides multiple data preprocessing tasks so that the user can choose. The user chooses a z-score normalization task. The task is inserted in the workflow in the position selected by the user.

Use case 1.2.4 Visualise the output of a (data preprocessing) z-score normalisation task (M)

An authenticated user wants to visualise the output from a z-score normalisation task after it has been executed in a workflow. The workflow includes a z-score normalisation task and it has executed successfully. The user sends a request to visualise the task outputs. The system outputs the datasets where the operation was applied. The user can visualise the normalised datasets, the normalised attributes and the object to apply normalisation to new data.
The things that could go wrong are:

- The z-score normalisation task did not execute successfully. The user sends a request to visualise the task outputs. The system outputs a message with the reason for the task failure.

Use case 1.2.5 Insert a (data preprocessing) one-hot encoding task on the workflow (M)
An authenticated user wants to insert a one-hot encoding task on the workflow in order to convert discrete data to a numerical format to use in the workflow. The workflow is already initiated. The user sends a request to insert a data preprocessing task on the workflow. The system provides multiple data preprocessing tasks so that the user can choose. The user chooses a one-hot encoding task. The task is inserted in the workflow in the position selected by the user. The user selects the attributes that must be encoded and the attributes that must be excluded from the encoding operation.

Use case 1.2.6 Visualise the output of a (data preprocessing) one-hot encoding task (M)
An authenticated user wants to visualise the output from a one-hot encoding task after it has been executed in a workflow. The workflow includes a one-hot encoding task and it has executed successfully. The user sends a request to visualise the task outputs. The system outputs the datasets where the operation was applied. The user can visualise the encoded datasets and the labels included in the encoded datasets.
The things that could go wrong are:

- The one-hot encoding task did not execute successfully. The user sends a request to visualise the task outputs. The system outputs a message with the reason for the task failure.

Use case 1.2.7. Insert a (data preprocessing) discretisation task on the workflow (S)
An authenticated user wants to insert a discretisation task on the workflow in order to have only discrete data to use in the workflow. The workflow is already initiated. The user sends a request to insert a data preprocessing task on the workflow. The system provides multiple data preprocessing tasks so that the user can choose. The user chooses a discretisation task. The task is inserted in the workflow in the position selected by the user.

Use case 1.2.8 Visualise the output of a (data preprocessing) discretisation task (S)
An authenticated user wants to visualise the output from a discretisation task after it has been executed in a workflow. The workflow includes a discretisation task and it has executed successfully. The user sends a request to visualise the task outputs. The system outputs the datasets where the operation was applied. The user can visualise the datasets.

Use case 1.3 Insert a feature selection task on the workflow (M)
An authenticated user wants to insert a feature selection task in the workflow in order

to know what are the best features present in the data and to select the most relevant features. The workflow is already initiated. The user sends a request to insert a feature selection task on the workflow. The system provides multiple feature selection tasks for the user to choose. The user chooses one of the available feature selection tasks that is then inserted in the workflow in the position selected by the user.

Use case 1.3.1. Insert (feature selection) Relieff algorithm on the workflow (M)
An authenticated user wants to insert the Relieff algorithm in the workflow in order to visualise the relevance of the features of a dataset being processed on the workflow and to select the most relevant features. The workflow is already initiated. The user sends a request to insert a feature selection task on the workflow. The system provides multiple feature selection tasks for the user to choose. The user chooses the Relieff task that is then inserted in the workflow in the position selected by the user. The user inputs values that are relevant for the application of Relieff algorithm. The user specifies a threshold that is used by the system to remove features from the dataset with score below the threshold. The user specifies a number of best features to be selected according to the ranking produced by the algorithm.

Use case 1.3.2 Visualise the output of a (feature selection) Relieff task (M)
An authenticated user wants to visualise the output from the application of a feature selection task that uses the Relieff algorithm after it has been executed in a workflow. The workflow includes (feature selection) Relieff task and it has executed successfully. The user sends a request to visualise the task outputs. The system outputs the ranking and scores produced by Relieff, the number of selected features and the threshold used.
The things that could go wrong are:

- The Relieff task did not execute successfully. The user sends a request to visualise the task outputs. The system outputs a message with the reason for the task failure.

Use case 1.3.3 Insert (feature selection) Info Gain algorithm on the workflow (M)
An authenticated user wants to insert the Info Gain algorithm in the workflow in order to visualise the relevance of the features of a dataset being processed on the workflow and to select the most relevant features. The workflow is already initiated. The user sends a request to insert a feature selection task on the workflow. The system provides multiple feature selection tasks for the user to choose. The user chooses the Info Gain task that is then inserted in the workflow in the position selected by the user. The user inputs values that are relevant for the application of Info Gain algorithm. The user specifies a threshold that is used by the system to remove features from the dataset with information gain below the threshold. The user specifies a number of best features to be selected according to the ranking produced by the algorithm.

Use case 1.3.4 Visualise the output of a (feature selection) Info Gain task (M)
An authenticated user wants to visualise the output from the application of a feature selection task that uses the Info Gain algorithm after it has been executed in a workflow. The workflow includes (feature selection) Info Gain task and it has executed successfully. The user sends a request to visualise the task outputs. The system outputs the ranking and the computed information gain, the number of selected features and the threshold used.
The things that could go wrong are:

- The Info Gain task did not execute successfully. The user sends a request to visualise

the task outputs. The system outputs a message with the reason for the task failure.

Use case 1.4. Insert a dimensionality reduction task on the workflow (S)
An authenticated user wants to insert a dimensionality reduction task in the workflow in order to reduce the dimensionality of the dataset used in the workflow. The workflow is already initiated. The user sends a request to insert a dimensionality reduction task on the workflow. The system provides multiple dimensionality reduction tasks so that the user can choose. The user chooses one of the available dimensionality reduction tasks that is then inserted in the workflow in the position selected by the user.

Use case 1.4.1. Insert (dimensionality reduction) principal component analysis (PCA) task on the workflow (S)
An authenticated user wants to insert a PCA task in the workflow in order to reduce the dimensionality of the dataset used in the workflow. The workflow is already initiated. The user sends a request to insert a dimensionality reduction task on the workflow. The system provides multiple dimensionality reduction tasks so that the user can choose. The user chooses the PCA task that is then inserted in the workflow in the position selected by the user. The user can input values to be used by the PCA algorithm.

Use case 1.5. Insert a model creation task on the workflow (M)
An authenticated user wants to insert a model creation task in the workflow in order to create a classification model trained with the dataset used in the workflow. The workflow is already initiated. The user sends a request to insert a model creation task on the workflow. The system provides multiple tasks to create a classification model using functional, tree, rule, lazy or bayesian algorithms for the user to choose. The user chooses one of the available algorithms for the creation of the model that is then inserted in the workflow in the position selected by the user.

Use case 1.5.1. Insert a task on the workflow to create a classification model using a functional SVM algorithm (M)
An authenticated user wants to insert a model creation task in the workflow in order to create a classification model trained with the dataset used in the workflow. The workflow is already initiated. The user sends a request to insert a model creation task on the workflow. The system provides multiple tasks to create a classification model using functional, tree, lazy or bayesian algorithms for the user to choose. The user chooses the functional SVM algorithm to create the model that is then inserted in the workflow in the position selected by the user. The user can input the parameter to be used by the SVM algorithm (C, kernel and the parameters of the kernel).

Use case 1.5.2. Visualise the output of a model creation task where the SVM algorithm was used (M)
An authenticated user wants to visualise the output of a model creation task where the SVM algorithm was used after it has been executed in a workflow. The workflow includes a model creation task where the SVM algorithm was used and it has executed successfully. The user sends a request to visualise the task outputs. The system outputs the created model for download, the parameters used to create the model and the predicted and expected results produced after testing the model.
The things that could go wrong are:

- The SVM task did not execute successfully. The user sends a request to visualise the task outputs. The system outputs a message with the reason for the task failure.

Use case 1.5.3. Insert a task on the workflow to create a classifier of the type tree using the CART algorithm (M)

An authenticated user wants to insert a model creation task in the workflow in order to create a classification model trained with the dataset used in the workflow. The workflow is already initiated. The user sends a request to insert a model creation task on the workflow. The system provides multiple tasks to create a classification model using functional, tree, lazy or bayesian algorithms for the user to choose. The user chooses the CART algorithm to create the model that is then inserted in the workflow in the position selected by the user. The user can input the parameter to be used by the CART algorithm.

Use case 1.5.4 Visualise the output of a model creation task where the CART algorithm was used (M)
An authenticated user wants to visualise the output of a model creation task where the CART algorithm was used after it has been executed in a workflow. The workflow includes a model creation task where the CART algorithm was used and it has executed successfully. The user sends a request to visualise the task outputs. The system outputs the created model for download, a tree with the representation of the model, the parameters used to create the model and the predicted and expected results produced after testing the model.

- The CART task did not execute successfully. The user sends a request to visualise the task outputs. The system outputs a message with the reason for the task failure.

Use case 1.5.5. Insert a task on the workflow to create a classifier of the type lazy using the K nearest neighbors algorithm (M)
An authenticated user wants to insert a model creation task in the workflow in order to create a classification model trained with the dataset used in the workflow. The workflow is already initiated. The user sends a request to insert a model creation task on the workflow. The system provides multiple tasks to create a classification model using functional, tree, lazy or bayesian algorithms for the user to choose. The user chooses the K nearest neighbors algorithm to create the model that is then inserted in the workflow in the position selected by the user. The user can input the parameter K to be used by the K nearest neighbours algorithm.

Use case 1.5.6 Visualise the output of a model creation task where the K nearest neighbors algorithm was used (M)
An authenticated user wants to visualise the output of a model creation task where the K nearest neighbors algorithm was used after it has been executed in a workflow. The workflow includes a model creation task where the K nearest neighbors algorithm was used and it has executed successfully. The user sends a request to visualise the task outputs. The system outputs the created model for download, the parameters used to create the model and the predicted and expected results produced after testing the model.
The things that could go wrong are:

- The k nearest neighbors task did not execute successfully. The user sends a request to visualise the task outputs. The system outputs a message with the reason for the task failure.

Use case 1.5.7. Insert a task on the workflow to create a classifier of the type bayesian using the Gaussian Naïve Bayes algorithm (M)
An authenticated user wants to insert a model creation task in the workflow in order to create a classification model trained with the dataset used in the workflow. The workflow is already initiated. The user sends a request to insert a model creation task on the workflow.

The system provides multiple tasks to create a classification model using functional, tree, lazy or bayesian algorithms for the user to choose. The user chooses the Gaussian Naïve Bayes algorithm to create the model that is then inserted in the workflow in the position selected by the user.

Use case 1.5.8 Visualise the output of a model creation task where the Gaussian Naïve Bayes algorithm was used (M)
An authenticated user wants to visualise the output of a model creation task where the Gaussian Naïve Bayes algorithm was used after it has been executed in a workflow. The workflow includes a model creation task where the Gaussian Naïve Bayes algorithm was used and it has executed successfully. The user sends a request to visualise the task outputs. The system outputs the created model for download, the parameters used to create the model and the predicted and expected results produced after testing the model.
The things that could go wrong are:

- The Gaussian Naive Bayes task did not execute successfully. The user sends a request to visualise the task outputs. The system outputs a message with the reason for the task failure.

Use case 1.5.9 Insert a task on the workflow to create a classifier of the type bayesian using the Multinomial Naïve Bayes algorithm (M)
An authenticated user wants to insert a model creation task in the workflow in order to create a classification model trained with the dataset used in the workflow. The workflow is already initiated. The user sends a request to insert a model creation task on the workflow. The system provides multiple tasks to create a classification model using functional, tree, lazy or bayesian algorithms for the user to choose. The user chooses the Multinomial Naïve Bayes algorithm to create the model that is then inserted in the workflow in the position selected by the user.

Use case 1.5.10 Visualise the output of a model creation task where the Multinomial Naïve Bayes algorithm was used (M)
An authenticated user wants to visualise the output of a model creation task where the Multinomial Naïve Bayes algorithm was used after it has been executed in a workflow. The workflow includes a model creation task where the Multinomial Naïve Bayes algorithm was used and it has executed successfully. The user sends a request to visualise the task outputs. The system outputs the created model for download, the parameters used to create the model and the predicted and expected results produced after testing the model.
The things that could go wrong are:

- The Multinomial Naive Bayes task did not execute successfully. The user sends a request to visualise the task outputs. The system outputs a message with the reason for the task failure.

Use case 1.6 Insert a task on the workflow to validate the model being constructed using different procedures (M)
An authenticated user wants to insert a validation procedure task in the workflow in order to specify the procedure that must be used while executing the tasks that compose the workflow. The workflow is already initiated. The user sends a request to insert a validation procedure task on the workflow. The system provides multiple validation procedure tasks for the user to choose. The user chooses one of the available validation procedures. The system inserts that task in the workflow in the position selected by the user.

Use case 1.6.1. Insert the cross-validation procedure to validate the model constructed in the workflow (M)
An authenticated user wants to insert a validation procedure task in the workflow in order to specify the procedure that must be used while executing the tasks that compose the workflow. The workflow is already initiated. The user sends a request to insert a validation procedure task on the workflow. The system provides multiple validation procedure tasks for the user to choose. The user chooses the k-fold cross-validation procedure. The system inserts the task in the workflow in the position selected by the user. The user inserts the number of repetitions to perform the cross-validation. The user selects nested or normal cross-validation. The user selects if the data in the cross-validation folds must be stratified. The user specifies if the data must be shuffled prior to the creation of the folds. The user selects an integer number to be used as a seed while shuffling and stratifying the data.

Use case 1.6.2 Visualise the datasets used in a cross-validation procedure (M)
An authenticated user wants to visualise the datasets used in the different folds of an executed workflow. The workflow includes a cross-validation procedure task and it has executed successfully. The user sends a request to visualise the task outputs. The system outputs the training and test datasets per fold used in the workflow. The user can visualise the datasets.
The things that could go wrong are:

- The cross-validation procedure task did not execute successfully. The user sends a request to visualise the task outputs. The system outputs a message with the reason for the task failure.

Use case 1.6.3. Insert the hold out procedure to validate the model constructed in the workflow (M)
An authenticated user wants to insert a validation procedure task in the workflow in order to specify the procedure that must be used while executing the tasks that compose the workflow. The workflow is already initiated. The user sends a request to insert a validation procedure task on the workflow. The system provides multiple validation procedure tasks for the user to choose. The user chooses the hold out procedure. The system inserts the task in the workflow in the position selected by the user. The inserts the proportion of data to use in the training and test partitions. The user specifies if the data in the partitions must be stratified. The user specifies if the data must be shuffled prior to the creation of the partitions. The user selects an integer number to be used as a seed while shuffling and stratifying the data.

Use case 1.6.4 Visualise the datasets used in a hold out procedure (M)
An authenticated user wants to visualise the datasets used as training and test sets in an executed workflow. The workflow includes a hold out procedure task and it has executed successfully. The user sends a request to visualise the task outputs. The system outputs the training and test datasets used in the workflow. The user can visualise the datasets.
The things that could go wrong are:

- The hold out procedure task did not execute successfully. The user sends a request to visualise the task outputs. The system outputs a message with the reason for the task failure.

Use case 1.6.5 Insert the train-validation-test procedure to validate the model constructed in the workflow (M)
An authenticated user wants to insert a validation procedure task in the workflow in order

to specify the procedure that must be used while executing the tasks that compose the workflow. The workflow is already initiated. The user sends a request to insert a validation procedure task on the workflow. The system provides multiple validation procedure tasks for the user to choose. The user chooses the hold out procedure. The system inserts the task in the workflow in the position selected by the user. The user inserts the proportion of data to use in the training, validation and test partitions. The user specifies if the data in the partitions must be stratified. The user specifies if the data must be shuffled prior to the creation of the partitions. The user selects an integer number to be used as a seed while shuffling and stratifying the data.

Use case 1.6.6 Visualise the datasets used in a train-validation-test procedure (M)
An authenticated user wants to visualise the datasets used as training, validation and test sets in an executed workflow. The workflow includes a train-validation-test procedure task and it has executed successfully. The user sends a request to visualise the task outputs. The system outputs the training, validation and test datasets used in the workflow. The user can visualise the datasets.
The things that could go wrong are:

- The train-validation-test procedure task did not execute successfully. The user sends a request to visualise the task outputs. The system outputs a message with the reason for the task failure.

Use case 1.6.7 Insert the "use all data" procedure to construct tasks and validate the model constructed in the workflow using all data (M)
An authenticated user wants to insert a validation procedure task in the workflow in order to specify the procedure that must be used while executing the tasks that compose the workflow. The workflow is already initiated. The user sends a request to insert a validation procedure task on the workflow. The system provides multiple validation procedure tasks for the user to choose. The user chooses the "use all data" procedure. The system inserts the task in the workflow in the position selected by the user.

Use case 1.6.8 Select classification performance metrics (e.g., accuracy, precision, recall, f-measure) (M)
An authenticated user wants to specify a classification performance metric in order to verify the classification performance of the classifier created in the workflow. The workflow is already initiated. The user requests a task to select the metrics for evaluation of the classifier being created in the workflow. The system inserts the required task and presents at least the metrics accuracy, precision, recall and f-measure for the user to select. The user select one or more metrics.

Use case 1.6.9 Visualise the classification performance of a produced model (M)
An authenticated user wants to visualise the classification performance according to certain metrics after the execution of a workflow. The workflow includes a task where selected metrics were introduced and it has executed successfully. The user sends a request to visualise the task outputs. The system outputs the classification performance according to the metrics previously selected and also a confusion matrix.
The things that could go wrong are:

- The task with selected classification performance metrics did not execute successfully. The user sends a request to visualise the task outputs. The system outputs a message with the reason for the task failure.

Use case 1.6.10 Insert a feature selection task to build models using different number of features (M)
An authenticated user inserts a feature selection task to build models with different features in order to obtain the model with the best features according to a classification performance metric. The user specifies more than one value as the number of attributes to select inside the feature selection task. The user specifies the preferred classification performance metric to be used as the decider for the best model configuration.

Use case 1.6.11 Visualise the configuration of features that produced the best model (M)
An authenticated user wants to visualise the configuration of features that produced the best model after the execution of a workflow. The workflow includes a feature selection task with more than one value as the number of attributes to select. The user sends a request to visualise the features included in the final model. The system outputs the ranking of the features and the number of best selected features.
The things that could go wrong are:

- The feature selection task did not execute successfully. The user sends a request to visualise the task outputs. The system outputs a message with the reason for the task failure.

Use case 1.6.12 Insert a model creation task to build models using different parameters (M)
An authenticated user inserts a model creation task to build models with different parameters in order to obtain the model with the best parameters according to a classification performance metric. The user specifies more than one value in the parameters of the model creation task. The user specifies a preferred classification performance metric to be used as the decider for the best model configuration.

Use case 1.6.13 Visualise the configuration of parameters that produced the best model (M)
An authenticated user wants to visualise the configuration parameters that produced the best model after the execution of a workflow. The workflow includes a model creation task with multiple optimizable parameters inserted. The user sends a request to visualise the parameters included in the final model. The systems outputs the parameters that produced the best model.
The things that could go wrong are:

- The model creation task did not execute successfully. The user sends a request to visualise the task outputs. The system outputs a message with the reason for the task failure.

Use case 1.7 Insert a numeric prediction task (regression) (S)
An authenticated user wants to insert a numeric prediction task in the workflow in order to create a regression model trained with the dataset used in the workflow. The workflow is already initiated. The user sends a request to insert a numeric prediction task on the workflow. The system provides multiple tasks to create a regression model using different regression algorithms. The user chooses one of the available algorithms for the creation of the model that is then inserted in the workflow in the position selected by the user.

Use case 1.8 Insert a clustering task (S)
An authenticated user wants to insert a clustering task in the workflow in order to create

a clustering model trained with the dataset used in the workflow. The workflow is already initiated. The user sends a request to insert a clustering task on the workflow. The system provides multiple tasks to create the clustering model using different clustering algorithms. The user chooses one of the available algorithms for the creation of the model that is then inserted in the workflow in the position selected by the user.

Use case 1.9 Insert an association learning task (S)
An authenticated user wants to insert an association learning task in the workflow in order to visualize the association between the features in the dataset used in the workflow. The workflow is already initiated. The user sends a request to insert an association learning task on the workflow. The system provides multiple tasks to make association learning using different association learning algorithms. The user chooses one of the available algorithms for association learning that is then inserted in the workflow in the position selected by the user.

Use case 1.10 Use a model produced after executing a workflow (S)
An authenticated user wants to use a model previously created in a workflow in order to make predictions using a new dataset. The user has access to the model. The user sends a request to use the model with the new dataset. The system predicts the outputs for the received dataset using the specified model and returns the results to the user. The user visualises the produced results.

Use case 2.0. Authenticate users (M)
A user wants to be authenticated in order to use the system. The user opens the GUI. The system displays a form for the user to enter his/her username and password. The user enters his/her username and password. The user sends a requests to the system to login. The system validates the entered username and password. The user is now authenticated. The things that could go wrong are:

- The user entered an invalid username and/or password, the system displays an error message.

Use case 2.1. Allow registration for unauthenticated users (M)
A user wants to be registered in order to access the system. The user opens the GUI. The system displays a form for the user to enter his/her username and password. The user enters his/her username and password. The user sends a requests to the system to register. The system registers the user. The user is now registered and authenticated.
The things that could go wrong are:

- The user entered an username that was already taken, the system displays an error message.

Use case 2.2. Create a new workflow (M)
An authenticated user wants to create a new workflow. The user sends a requests to the system to create a new workflow. The system asks the user to insert a name for the workflow. The user inserts a name for the workflow. The system creates a new workflow with that name and displays it so that the user can add tasks to it.

Use case 2.3. Save workflow (M)
An authenticated user wants to save the workflow that he is currently working on. The user sends a request to the system to save the workflow. The system saves the workflow and displays a message reporting that the workflow was saved.

Use case 2.4. Open workflow (M)
An authenticated user wants to open a workflow to visualise, change or execute it. The system displays a list of his saved workflows. The user sends a request to the system to open the chosen workflow. The system retrieves and displays the workflow to the user.

Use case 2.5. Execute workflow (M)
An authenticated user wants to execute a workflow to visualise the results. The workflow tasks are inserted in a correct order. The user sends a requests to the system to execute the workflow. The system validates the inputs and the order of the tasks; executes the workflow and displays the execution progress. When the execution concludes, the system displays the outputs of each task.
The things that could go wrong are:

- The user executed a workflow that is invalid (tasks in an invalid order or invalid inputs), the system displays an error message.
- An unexpected error occurred during the execution of a task, the system displays an error message.

Use case 2.6. Stop workflow (M)
An authenticated user wants to stop the workflow during its execution. The workflow that the user wants to stop is opened. The user sends a requests to the system to stop the execution of the workflow. The system stops the execution of the workflow and displays a message reporting that the workflow was stopped.

Use case 2.7. Delete workflow (M)
An authenticated user wants to delete a workflow from his saved workflows. The system displays a list of his saved workflows. The user sends a request to the system to delete a given workflow. The system removes all traces of that workflow from the system. The system displays a message reporting that the workflow was deleted.

Use case 2.8.1 Upload a csv dataset (M)
An authenticated user wants to upload a dataset. The user chooses a dataset with the csv format to upload and specifies if it has headers or not. The user uploads the dataset to the system. The system receives the dataset and displays a message reporting that the dataset has been saved, and gives an identifier of the dataset to the user for future access.
The things that could go wrong are:

- The upload of the dataset failed. The system displays an error message.
- The dataset is in an invalid format or corrupted. The system displays an error message.

Use case 2.8.2 Upload an arff dataset (S)
An authenticated user wants to upload a dataset. The user chooses a dataset with the arff format. The user uploads the dataset to the system. The system receives the dataset, displays a message reporting that the dataset has been saved, and gives an identifier of the dataset to the user for future access.
The things that could go wrong are:

- The upload of the dataset failed. The system displays an error message.
- The dataset is in an invalid format or corrupted. The system displays an error message.

Use case 2.9. Delete a dataset (M)
An authenticated user wants to delete a dataset from his saved datasets. The system displays a list with his/her uploaded datasets. The user sends a request to the system to delete a given dataset. The system removes all traces of that dataset. The system displays a message reporting that the dataset was deleted.

Use case 3.0. Open a workflow template (M)
An authenticated user wants to open a workflow template. The system displays a list of templates created by the system. The user sends a request to the system to open a given template. The system retrieves and displays the template.

Use case 3.2. Remove a task from the workflow (M)
An authenticated user wants to remove a task from the workflow. The user sends a request to the system to remove a task from the workflow. The system removes that task.

Use case 3.3. Enable the definition of ranges in parameters of tasks that allow more than one value (C)
An authenticated user wants to specify a range of numbers for a task parameter that allows the definition of lists of numbers. The user selects the parameter and specifies the start and end values of the range. The system interprets the range as distinct numbers.

3.3 Quality Attributes

A system can not succeed by delivering functionalities in isolation. These functionalities must be delivered with certain qualities associated. Thus, a system under construction must be concerned not only with functionalities but also with quality attributes that must be considered through design, implementation and deployment. In this section we will provide a utility tree to describe scenarios for each quality attribute, followed by the strategies that were used to satisfy each one.

3.3.1 Utility Tree

Quality attributes to be addressed by a system must not be vague. Just specifying that the system must be available, or that it must be secure or maintainable is not enough. To identify the architectural significant requirements (ASR) we made use of the quality attribute scenarios. A quality attribute scenario represents a situation that might happen in the system, in which a specific quality attribute must be verified according to a certain measure.
It consists of six parts:

- **Source of stimulus**: This is the entity that generated the stimulus.
- **Stimulus**: A condition that must be taken to consideration when it is received by the system.
- **Environment**: The state of the system or other component that is being stimulated.
- **Artefact**: The component that is stimulated;
- **Response**: The activity undertaken after the arrival of the stimulus.
- **Response measure**: Measure of the response so that the requirement can be tested.

To specify the main quality attributes to be addressed and their scenarios we present the following utility tree (Table 3.1).

Utility trees help to concretise and prioritise quality goals [29]. High-level quality attributes (e.g., Availability, security, performance) are presented at the first level, that in the case of Table 3.1 appear at the first column. These high-level quality attributes are then refined in greater detail in level two. The final and third level are the quality attribute scenarios.

The prioritisation of the utility tree is done along two dimensions: by the importance of each ASR to the success of the system and the degree of perceived risk posed by its achievement. The prioritisation is done using relative ranking **High (H), Mid (M), Low (L)**. For example an attribute with ranking (H, M) means that the attribute has high importance to the success of the system and medium risk to achieve.

Quality attribute	**Attribute characterization**	**Attribute scenarios (ASR)**
QA-1. Modularity	Separation of each service in clear and well defined modules	AS-1: A system developer introduces a new service to the system, during system development. The new service is deployed in the system and will contain everything necessary to execute its functionalities. (H, L)
QA-2. Maintainability	Capacity to adapt to new functionalities	AS-2.1: A system developer introduces a new data science service in the system during system development. The new service is deployed in the system and becomes available to be used by other system components after registration without involving changes in more than one other service. (H, H)
	Capacity of making changes in a service without changing the remaining system	AS-2.2: A system developer changes the functionalities of a service during system development, maintaining its interface. The changed service is deployed without requiring any modification in the remaining system. (M, M)
QA-3. Interoperability	Accessibility from external services	AS-3.1: An external system makes a request for the execution of a data science workflow, while the system is in normal operation. The system is accessible through its well known and documented interface and executes the requested workflow. (L, M)
	Capacity of communication between different services	AS-3.2: A system developer creates a microservice during system development. The new microservice becomes available in the system and can communicate with other services through technology agnostic communication protocols (e.g., HTTP). (M, H)

QA-4. Usability	Satisfaction	AS-4.1: An authenticated user or external system requests the execution of a data science workflow while the system is in normal operation. The system starts the execution of the workflow and keeps information about the execution of every task that compose the workflow, so that information about the progress of the execution of a workflow can be returned to the caller. (H, L)
	Efficiency	AS-4.3: An authenticated user inserts a data science task on a workflow, while the system is in normal operation. The system inserts the task on the workflow with default values on every parameter. (H, L)
		AS-4.4: An authenticated user creates a workflow to produce models with different features and parameters to have access to the best model configuration, while the system is in normal operation. The system executes the workflow, creating models with different features and parameters and returns the best model to the user, without requiring the user to create multiple workflows with the different features and parameters to produce the best model configuration. (H,M)
	Effectiveness	AS-4.5: An authenticated user has a specific goal that is achieved through a data science task. The user adds a task to the workflow and executes it, while the system is in normal operation. The task was configured properly according to the users' intentions and the system displays the expected results at least 80% of the times. (H, L)
	Learnability	AS-4.2: A user executes an operation in the system, while the system is in normal operation. S/he will not have difficulties learning how to use the interface or finding the required functionalities. (H, M)
QA-5. Reliability	Successful job execution	AS-5: A data science service crashes while executing a task from a data science workflow when the system is in normal operation. The system retries the execution of the task in another service without any necessary intervention from the user and without losing the results of previously computed tasks. (H, H)

QA-6. Availability	System uptime	AS-6: A system service crashes while the system is in normal operation. The system detects the crash and heals the service maintaining the functionalities of every service available 99.9% of the time. (M, H)
QA-7. Performance	Execution speed of data science tasks	AS-7: A data science task arrives to the system to be processed while the system is in normal operation. The task is executed by the system with a performance comparable to running the same task on a computer with 8 GB of RAM and a CPU with 4 cores at 3GHz. (H, M)
QA-8. Scalability	Ability to maintain the performance independently of the number of requests	AS-8: A user places two times more requests than what is expected by the system during normal operation. The system processes this load without loss of performance. (H, H)
QA-9. Elasticity	Ability to use resources efficiently	AS-9.1: The system verifies that some modules/services are consuming memory or CPU below the average, while the system is in normal operation. The system gradually removes the modules/services that are consuming memory or CPU below the average. (M, H)
		AS-9.2: The system verifies that some modules/services are consuming memory or CPU above the average, while the system is in normal operation. The system gradually increases the modules/services that are consuming memory or CPU above the average. (M, H)
QA-10 Security	Confidentiality	AS-10.1: An authenticated user inserts a dataset in the system, while the system is in normal operation. The system receives the dataset and makes the dataset accessible only by authorized users. (H, M)
		AS-10.2: An unauthorised user inserts a wrong username or password to enter in the system, while the system is in normal operation. The system denies the entrance to the user without exposing any data associated with the inserted username. (H, M)

Table 3.1: Quality attributes' utility tree

3.3.2 Modularity

Modularity is the practice of encapsulating portions of the application into self-contained services that can be individually designed, developed, tested, and deployed with little or no dependency on other components or services [18].
The strategies used to achieve modularity were:

- Creating a system following a service oriented architecture, more precisely a microservices one, where each component is designed independently and with clearly defined boundaries.

This strategy and the containerisation of each microservice allowed us to ensure the modularity of the system.

3.3.3 Maintainability

Maintainability is the aptitude of a system to undergo repair and evolution [6]. By designing a maintainable system we can update it faster and with lower costs. Maintainable software can be reused, thus alleviating costly update time. Also, any faults found in the software can be easily diagnosed and corrected, reducing downtime and meeting delivery schedules. Software maintainability can also improve system availability by reducing the downtime [35]. Data scientists might require multiple machine learning algorithms to accomplish their tasks. Due to the innovation that is associated with the machine learning field, the system should be specially prepared to support new machine learning functionalities, and easy reparations in the deployed functionalities.
Inspired in the book "Software Architecture in Practice" by Len Bass, et al. [7], the strategies used to achieve maintainability were :

- **Early planning**: anticipating what and how programs might be modified at a later stage. It was visible from the beginning that the services that would go more under changes would be the ones responsible for exposing machine learning algorithms.
- **Modular design**: defining subsets and simplifying functionality. Most of the functionalities in the current architecture are split to different microservices.
- **Object-oriented design**: encapsulating both methods and data structures to achieve a higher level of independence than that of modular design. Object-oriented design was specially explored in the workflows service where there is high complexity in operations related to the conversion of data science workflows received from user to their system representations.
- **Coding Standards**: by following coding standards we improve the readability of our code. This allows engineers that are not familiar with the code to understand it more quickly and thoroughly, making the software more easy to maintain.

3.3.4 Interoperability

Interoperability is related with the way that each service is capable of communicating with each other through completely understood interfaces, regardless of programming languages used for implementation. This is very important because in a large system, such as this

one, there will be teams focused in different microservices. One team do not have to know the inner workings of a service produced by another team, they just need a standard interface that gives access to the functionalities exposed by the service.
The strategies used to achieve interoperability were:

- Each service has an interface that provides its functionalities by using a uniform and predefined set of stateless operations available through HTTP requests. The standard adopted was the REpresentational State Transfer (REST). There are many standards such as 'Simple Object Access Protocol (SOAP)' or 'Remote Procedure Calls (RPC)'. The REST standard was chosen because it scales very well, since the client and server are very loosely coupled. With REST, the server is free to change the exposed resources at will. In contrast, with SOAP and in certain RPC schemes, the client and server must first agree on a detailed protocol that typically needs to be compiled in both ends. As a result of these factors, REST is a better choice, especially for services that undertake rapid evolution, as is the case of our system.

3.3.5 Usability

Usability is the degree to which a software can be used by specified consumers to achieve quantified objectives with effectiveness, efficiency, and satisfaction [1]; it also relates with how easy is for the user to work on the platform and achieve his goals; a system with a high learning curve or too complex is not desirable. It was based on the points mentioned above that the Grafical User Interface (GUI) was developed.
The strategies used to achieve usability were:

- Each task that can be inserted in a data science workflow comes with relevant parameter values guide the users in the process.
- When the user executes a workflow, the progress of that execution is shown to the user.
- Colours and icons were chosen according to common uses in other familiar applications. E.g colour green means success; the close or remove button is represented with the 'x' symbol.
- The addition of tasks to a workflow is an easy process that takes no more than 3 clicks.
- To allow the execution of workflows to select the best model according to different features and parameters, without requiring the user to create different workflows for each configuration of features and parameters. The system will parallelize the creation of models with different features and parameters on the background in order to present the best model to the user.
- As the user builds a workflow, s/he will only be shown the tasks that can be inserted next according to the machine learning process, guiding the user during the process. This is done to help the user achieve his goals, increasing effectiveness, efficiency, understandability and improving the users' satisfaction.

To further test and validate this attribute we conducted usability tests that can be seen in section 6.2.

3.3.6 Reliability

Reliability relates with the trust that the user puts on the system to perform a certain task and its correct execution.
To create a reliable system, the strategies used were:

- During the execution of a workflow there is a probability that a task in the workflow might not complete successfully. It can be because the service working on that task crashed, the network where that service is became inaccessible, or another unknown reason. With the objective of not failing the whole workflow, the system will reschedule each task that fails in another service, and the component responsible for that will be the orchestration service.

3.3.7 Availability

Availability is concerned with system failure and its associated consequences. A system failure occurs when the system no longer delivers a service consistent with its specification. If the platform is not available when the user needs it, it means that it is not fulfilling its functional requirements.
Inspired in the technical report "Realizing and Refining Architectural Tactics: Availability" by James Scott, et al. [40], the strategies used to achieve availability were:

- **Fault detection**: by **monitoring** the status and timeouts of each task, the orchestration service is able reschedule the failed or timedout tasks, increasing the availability and reliability of the system. Also, in the data science services the use of the library **Condu** allowed us to add an additional layer in **exception detection** by reporting the exceptions occurred to the orchestration service.
- **Fault recovery**: we employ redundancy of components, avoiding single points of failure. We do this by replicating each microservice.
- **Fault prevention**: if a service is deemed unhealthy by detecting certain unusual behaviours, we can put it out of service and restart it in order to scrub latent faults such as memory leaks.

3.3.8 Performance

Performance as a software quality attribute refers to the timeliness aspects of how software systems behave.
Inspired in the book "Software Architecture in Practice" by Len Bass, et al. [7], the strategies used to achieve performance were:

- **Run multiple requests concurrently**: Use multiple instances to process requests and use load balancers to balance the requests between the available instances.
- **Increasing available resources**: by using microservices, our system enables an easy increment of resources according to the functionalities that have higher demand.
- **Bound execution time**: limiting how much execution time can be used to respond to an event. In the case of the data science tasks the orchestrator enforces this by setting timeouts for each task.

We were successful in employing this strategies and compared the performance of the Data Science for Non-Programmers (DS4NP) platform with H2O, whose results can be seen in section 6.3. However, we could not test the ASR scenario specified for this attribute, due to time constraints.

3.3.9 Scalability

Scalability tells us about how can this architecture cope when its requirements increase in size, more specifically the ability of the system to maintain its performance when receiving an increasing number of requests.
The strategies used to achieve scalability were:

- We developed services with well defined boundaries that can be scaled independently according to their workload.
- The system was designed to be able to scale by adding more machines into the resource pool (horizontal scaling). Like in many systems, it is also possible to add more RAM or CPU power to existing machines, but, as the system expands scaling horizontally is more economically viable.

3.3.10 Elasticity

is the degree to which the system is able to adapt to workload changes by provisioning and de-provisioning resources in an autonomic manner, such that at each point in time the available resources match the current demand as closely as possible [26].
The strategies used to achieve elasticity were:

- The use of technologies used for managing containers such as Kubernetes and Docker to enable automatic elasticity of services according to the CPU and RAM that is consumed.

3.3.11 Security

The definition of a security attribute depends on the context in which the attribute is addressed. For this platform the security feature that we identified as fundamental was the confidentiality of the datasets. Many scientists work with data that is sensitive and must be protected from unauthorized users. Not satisfying this requirement would make the use of the system impracticable for many data scientists.
The strategies used to achieve security were:

- Every dataset is identifiable and only accessible having a token that is at least 16 characters. According to the paper "Advances of Password Cracking and Countermeasures in Computer Security" by Aaron L.-F. Han. et. al [21], brute forcing a password 16 characters (128 bits) long would take $5.4 * 10^{18}$ years with 106 000 decryptions per second. According to these results, we conclude that using tokens for the datasets with at least 16 characters, will provide more than enough protection.
- Introduce an authentication method for every exposed microservice to validate if the request originates from an authenticated user/external system.

3.4 Restrictions

The only restriction identified at the moment is cost. Nowadays, the **cost** of using data science services is high. Using Free and Open Source Software (FOSS) would drive the cost way down and make it affordable for the average user.

Chapter 4

Architecture

The architecture is a reflection of the functional and non-functional requirements previously identified. It was designed to ensure that all functional requirements can be implemented and the quality attributes satisfied. The design patterns adopted is what is commonly referred as a **Service Oriented Architecture (SOA)**, more precisely a **microservices** one. Choosing this pattern is a good choice, since its benefits [19] are inline with the quality attributes that were established for this project. Developing in this manner reduces many struggles of integrating each module of the system, eliminates restrictions such as programming languages, and provides us with better fault isolation.

The diagram for the architecture should have a focus on the components present in the system, how they interact and technology adoptions. After investigating standard ways of structuring the architecture we adopted the C4 model for its easy and hierarchical way of representing the architecture. The C4 model was created to help teams communicate how they are planning to build a software system (up-front design) or how an existing software system works (retrospective documentation and knowledge sharing). It has 4 levels of abstraction [9]:

- **Level 1** is the system context diagram. It shows how the system being built interacts with other systems or people.
- **Level 2**, is a container diagram. It zooms into the software system, and shows the containers (applications, data stores, microservices, etc...) that make up the software system. Technology decisions are also a key part of this diagram.
- **Level 3**, is a component diagram. It zooms into an individual container to show the components inside it. These components usually depict groupings of code used in the containers codebase.
- **Level 4**, the last one, represents the code. It is usually displayed with UML class diagrams.

Taking into account the requirements for the system and the investigation done of other data science applications we will now present the proposed architecture.

As seen in Figure 4.1 the system can be used by a data scientist or an external software system. The user interacts with a graphical interface that will allow him to send HTTP requests to the system; external systems will use the same HTTP requests to have access to the same exposed functionalities.

In the level 2 view of the architecture (Figures 4.2 and 4.3) we will be able to visualise all the databases and microservices present in the system. The system has many containers so this view is separated in two figures.

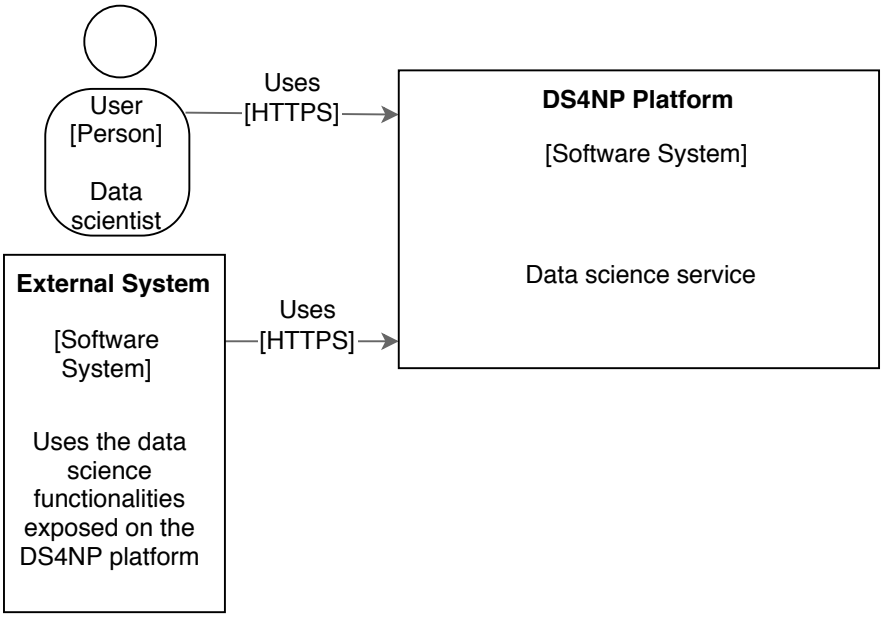

Figure 4.1: C4 model diagram - Level 1

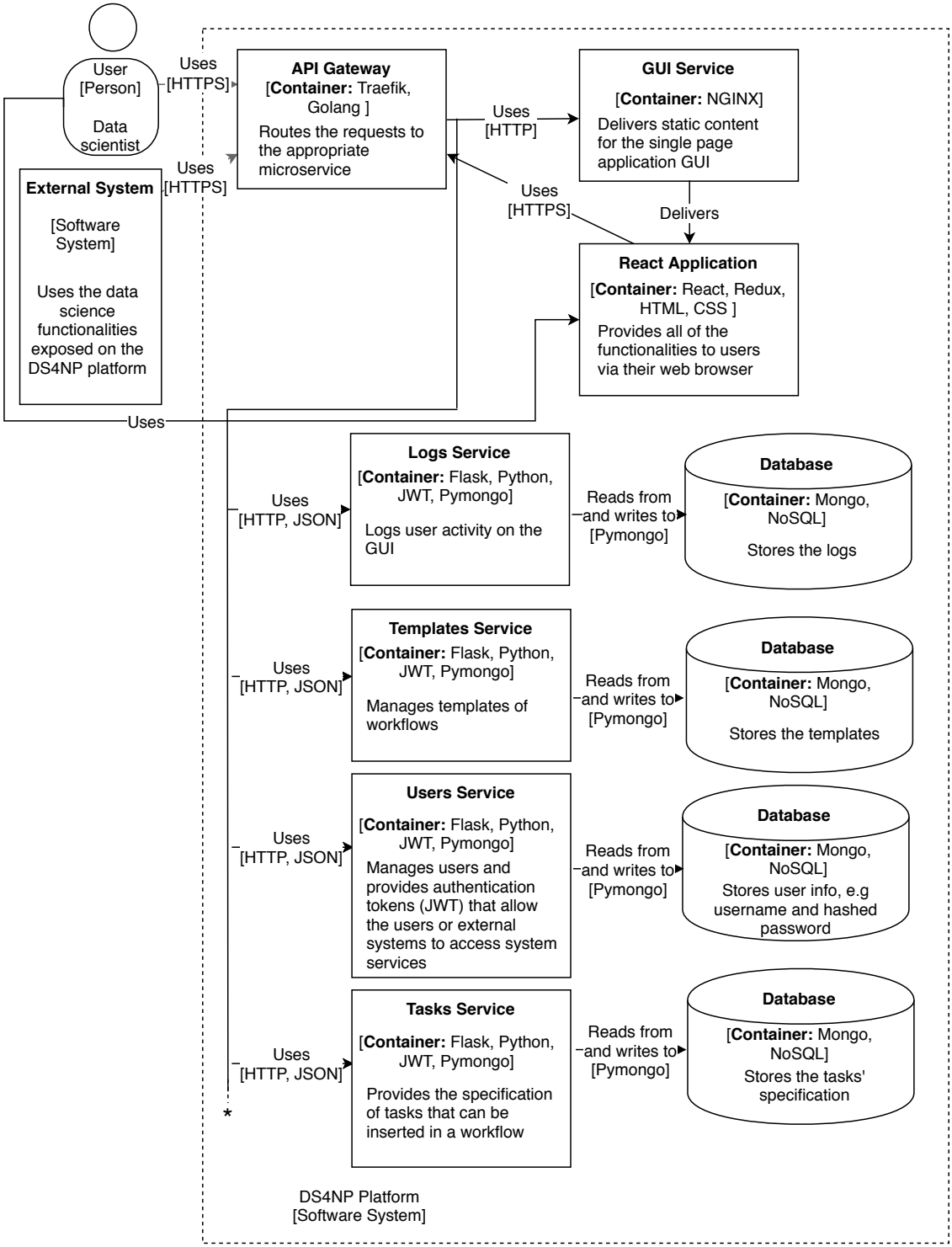

Figure 4.2: C4 model diagram - Level 2. Part 1

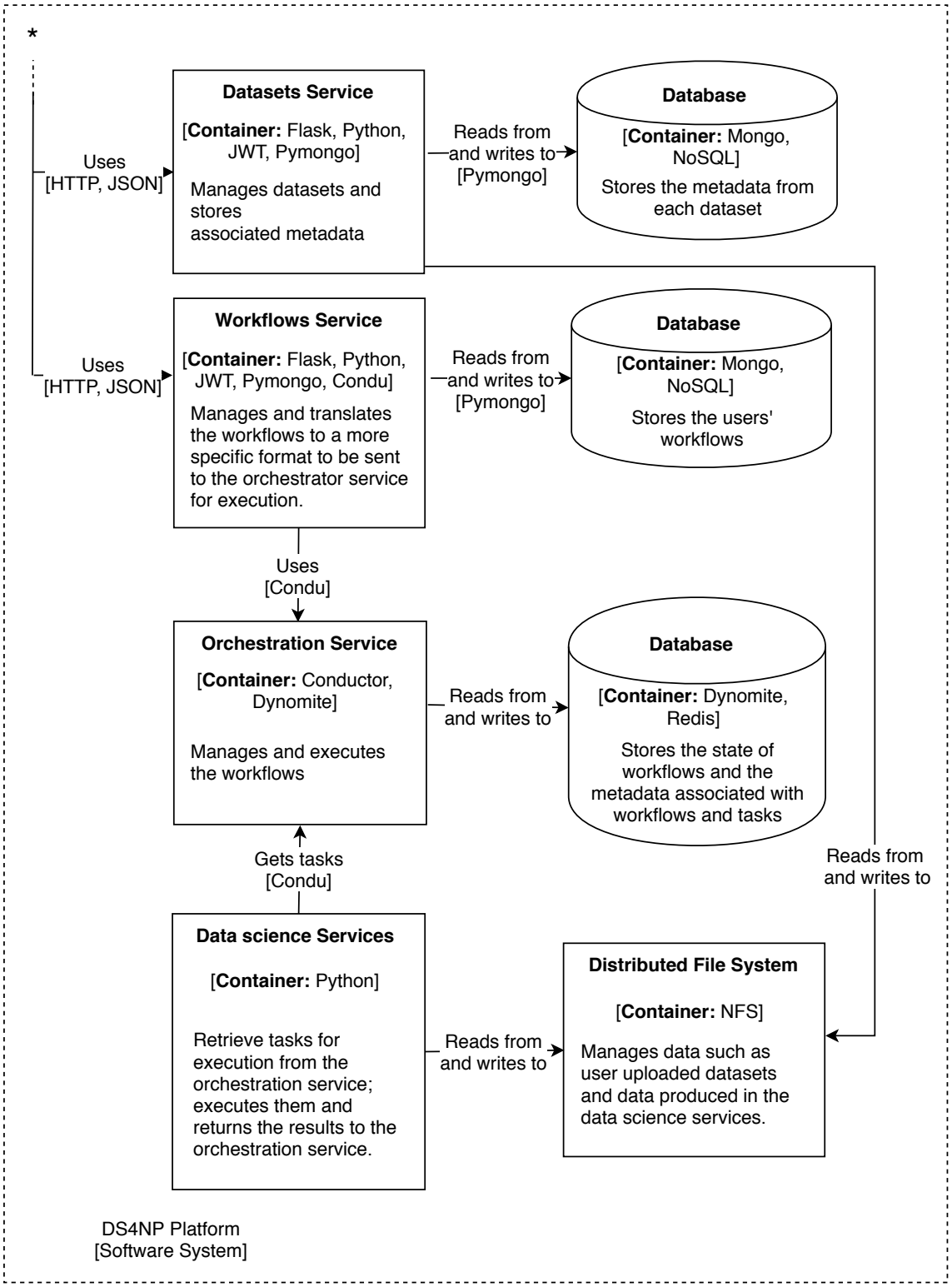

Figure 4.3: C4 model diagram - Level 2. Part 2

The **API gateway** serves as the entry point to the system and is responsible for routing every request to the appropriate microservice. The gateway decides where to send the request based on the path, e.g., if the request has '/index.html' for a path, the gateway redirects to the GUI service; the '/workflows' path would be redirected to the workflows service; '/logs' path would be redirected to the logs service and so on...

The **GUI service** delivers the graphical interface for the user; the service will send HTML and Javascript files that will run on the user's web browser. Further requests from the user to the system will originate from the web browser (**React application**) and will be sent to the API gateway.

The **Logs service** receives logs that are sent from the user's web browser. After processing a user request to the system (e.g., creating workflows, adding tasks, deleting datasets, etc...) the React application will send the result of the request (successful/unsuccessful) and relevant data to the Logs service. Confidential information such as datasets is not stored by the Logs service. The goal of this service is to provide data about user behaviour to the developers to help them improve the interface.

The **Templates Service** stores and provides predefined data science workflows. These workflows can be searched and accessed by the users to see examples of workflows.

The **Users service** allows the users or external systems to create accounts, and to authenticate themselves. Authentication is required to use all the other services accessible from the gateway except for the GUI service.

The **Task service** provides the description of data science tasks that can be used by users or external systems to build a data science workflow.

The **Datasets service** is used to store the datasets and its metadata (e.g., dataset's name and feature/class labels). The user account that owns each dataset is stored in the metadata database along with the metadata. The uploaded datasets are stored in the Distributed File System.

The **Workflows service** is responsible for managing the workflows created by users or external systems. It receives sequential data science workflows and translates them to a lower level format that enables the orchestration of data science microservices by the orchestration service. Each logical workflow is associated with its low-level representation, and is stored in a database so that it can be retrieved later to verify its execution status.

The **Orchestration service** manages the execution of low-level workflows translated in the workflows service. When the orchestration service receives a request from the workflows service to execute a workflow, the tasks that compose the workflow are put in a queue. Then, data science services will be able to pull scheduled tasks for execution. This service will hold the execution status of the workflows and the outputs of each processed task on the workflow. These results are requested from the workflows service on user request to update the status of sequential data science workflows.

The **Data science services** are in reality several microservices that execute specific tasks that compose the workflows present in the orchestration service. They contact the orchestration service to get specific scheduled tasks for execution. Upon receiving a task, they proceed to its execution and return the results to the orchestration service.

The **Distributed File system** is where large files like datasets are stored. It is accessed both by the datasets service and by the data science services. The datasets service uses the distributed file system to store user uploaded datasets or to retrieve user's files. On the other side, the data science services access the distributed file system not only to read datasets uploaded by users but also to store data created on the services or to access data created by other services, such as datasets, models and predictions.

For the purpose of describing the system architecture as a whole, only the level 1 and 2 figures of the C4 model are presented in this section. In the Implementation chapter (5) each service will be described in more detail and the level 3 view of the services shown.

This page is intentionally left blank.

Chapter 5

Implementation

This chapter describes the microservices that compose the Data Science for Non-Programmers (DS4NP) platform in more detail, focusing on the motives behind the technologies adopted and how they work.

It is important to notice that the services and components described in this chapter were developed by the candidate Bruno Lopes and the MSc Student, Artur Pedroso.

Artur Pedroso developed the following services:

- Data science services.
- Distributed file system.

The candidate Bruno Lopes developed the following services and the associated databases:

- API gateway
- GUI service
- React Application (Grafical User Interface (GUI))
- Logs service
- Templates ervice
- Users service
- Tasks service
- Orchestration service (Netflix Contuctor) and the Python library **Condu** to comunicate with the orchestration service.

In a joint effort, the following services were developed:

- In the workflows service, Artur was responsible for the 'translator' component and I developed the rest of the functionalities.
- In the datasets service, Artur was responsible for the management of datasets' and I was responsible for the metadata.

5.1 Microservices

As already described in the architecture chapter, the system follows a microservice oriented architecture that is composed of a collection of loosely coupled services. In the next subsections we will describe the components that compose each service and, for some that need to be described in more detail, we will show the level 3 view of the c4 model to help us better understand their inner workings.

5.1.1 API Gateway

In theory, the clients could make requests to each microservice directly. Each microservice would have a public endpoint (e.g., https://<service name>.api.ds4np.pt). This URL would map to the microservices' load balancer, which would distribute the requests across the available instances.

However, this approach would impose more complications and challenges. The first problem that is more apparent is the complexity of keeping track and calling each one of the many services that would be needed for each operation.

Another drawback with this approach is that refactoring the services would be a difficult task. Over time we might want to change the system structure and the way the functionalities are partitioned into services (e.g., we might merge two services or split a service into two or more services). If the clients communicate directly with the services, then performing this kind of refactoring can be extremely difficult.

Because of these kinds of problems it makes more sense to insert an API gateway in the system and not having the clients talk directly to the microservices.

The gateway used was Træfik [16]. There are other alternatives available such as Netflix Zuul, but Træfik has better documentation and is easier to configure, making it a better choice.

The routes are decided based on the path. There is a configuration file with all paths to be accepted and its destination. To illustrate, all the requests that have **/tasks** as path are sent to the Tasks service; a request with **/auth** is sent to the Users service, etc...

5.1.2 Graphical User Interface Service

This service job is to send the index.html and the JavaScript files to the users' web browser.

There are many web servers available that we could use to serve these files but the one that was adopted was the NGINX [36]. Because of its popularity and reliability it was the first one tried. After configuring it and seeing it in action, we considered it to be an adequate choice for the platform.

5.1.3 Logs Service

This service was first implemented because there was a need to analyse the users behaviour for the usability tests. The analysis allowed us to determine if they were doing the exercises correctly and to calculate the time required to complete each one. Having this goal achieved, the service is still useful for the system in production. Doing the analysis of the

users' behaviour is one of the tools, and a very important one, that the developers have available to improve the GUI.

Each log is represented with a json object and has the following fields:

- **Action**: is an unique string that identifies the action performed that produced the log.
- **Success**: boolean field (true or false) that tells us if the action was successful or not.
- **Data**: In this field we put data deemed relevant to the action, e.g., when an action such as uploading a dataset fails, we store the error message and when a user creates a workflow we store the name of the workflow that was created.
- **Time**: This field stores the date and time when the log object was created on the users' web browser before being sent to the logs service.

Figure 5.1 helps us visualise the components present in this service.

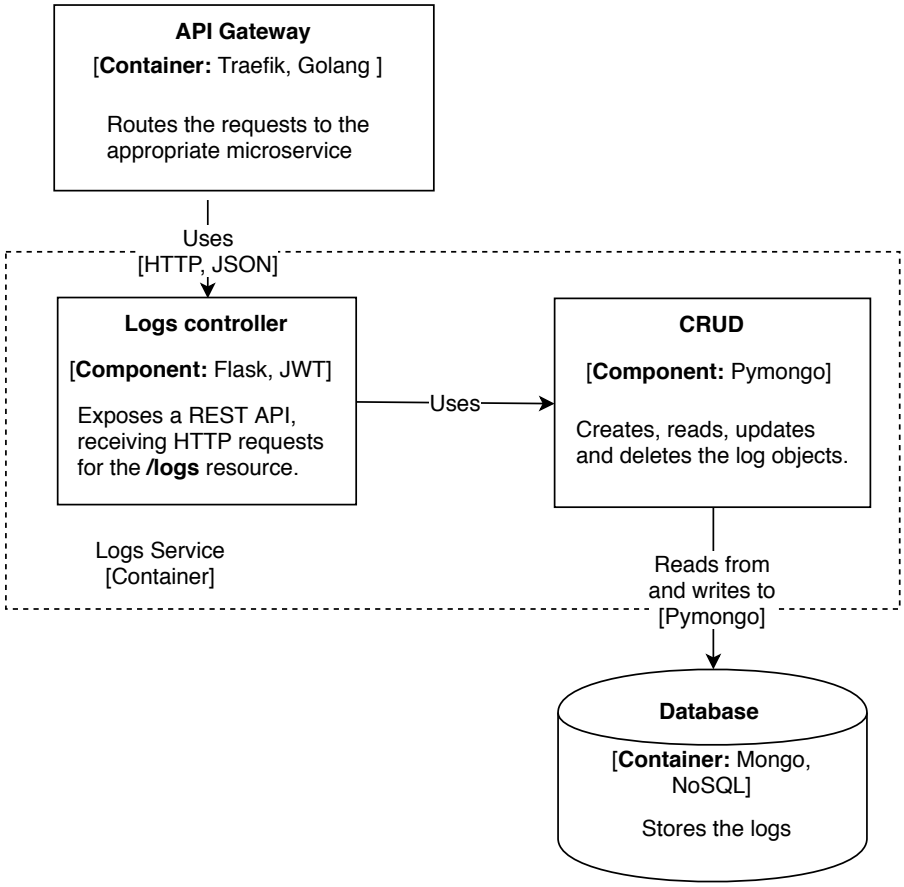

Figure 5.1: Logs service: level 3 view.

5.1.4 Templates Service

Besides storing the workflows created by the users, it is also important to provide them with example ones to show functionalities or help the users learn how to perform certain tasks. Each template has keywords associated with them to allow its search by the users. Even though this feature was implemented and tested in this service, it was not possible to add the search functionality on the GUI due to time constraints.

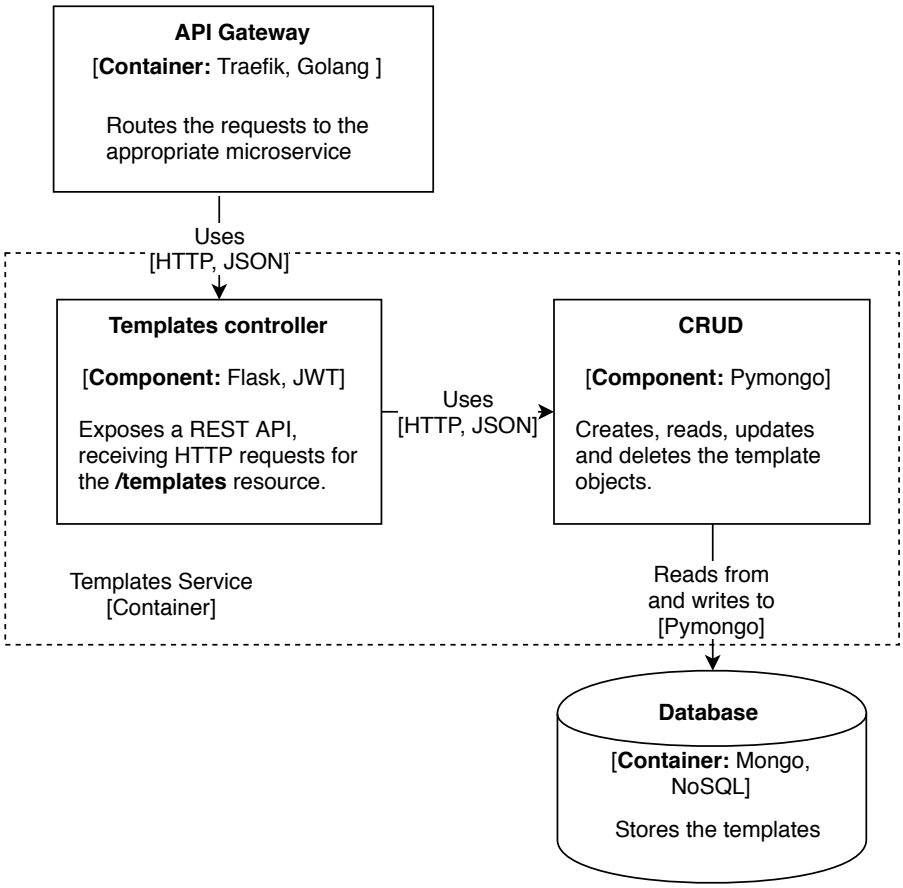

Figure 5.2: Templates service: level 3 view.

5.1.5 Users Service

This service is responsible with the management of accounts and authentication. The authentication method follows the internet standard reference RFC 7519 [27] named Json Web Token (JWT).

The authentication is conducted in the following steps: the client sends his username and password; the service verifies if there is a match in the database; if there is a match, a token is sent to the client securing its claim to access the system.

The **token** contains the username and a signature signed with a cryptographic key shared between the other services called '**secret**'. This secret allows other services in the system to verify the token and know the username of the client that sends each request without needing to read from the database. Going to the database for every request the system receives would cause an enormous overload on the users database. Another major advantage of using JWT is that we can introduce a time limit inside the tokens or any other

information deemed necessary for the authentication. This way, if an attacker catches a token he only has a brief window of time before that token becomes useless.

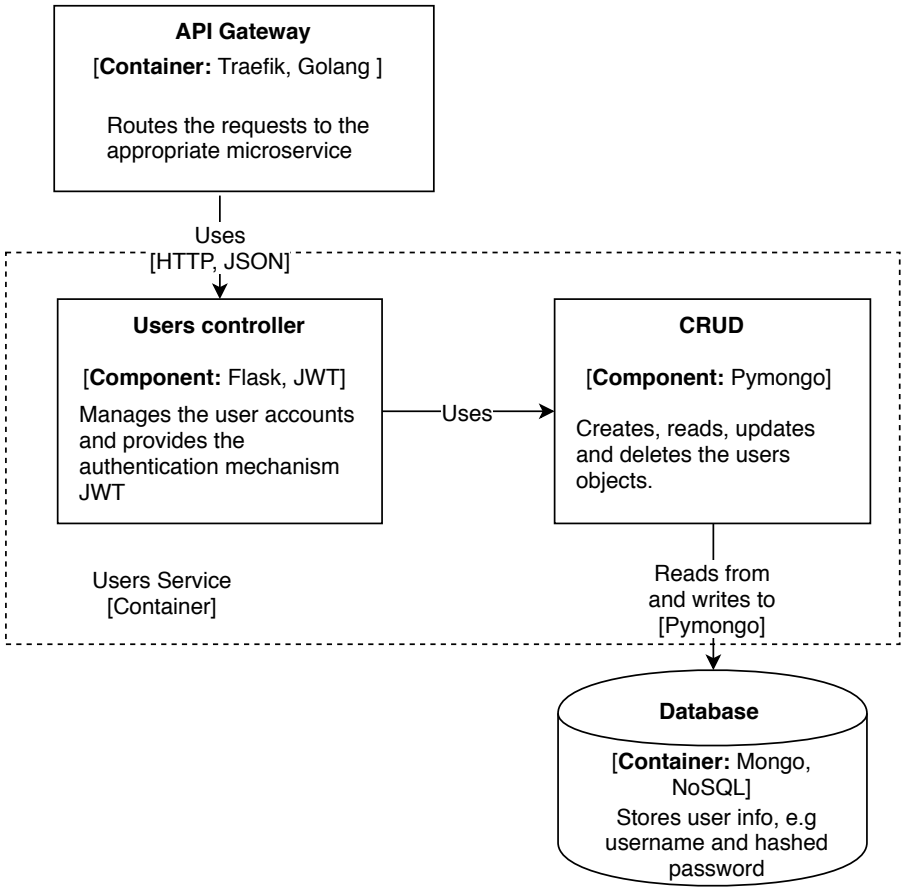

Figure 5.3: Users service: level 3 view.

5.1.6 Datasets Service

This service is responsible for managing the datasets uploaded by the users and sending them to the distributed file system. For each dataset the following information is stored: attributes, type of file (csv or arff), name, owner and an unique identifier; we called this information metadata. By using the metadata we are able to keep track of the users' datasets to allow their use in workflows.

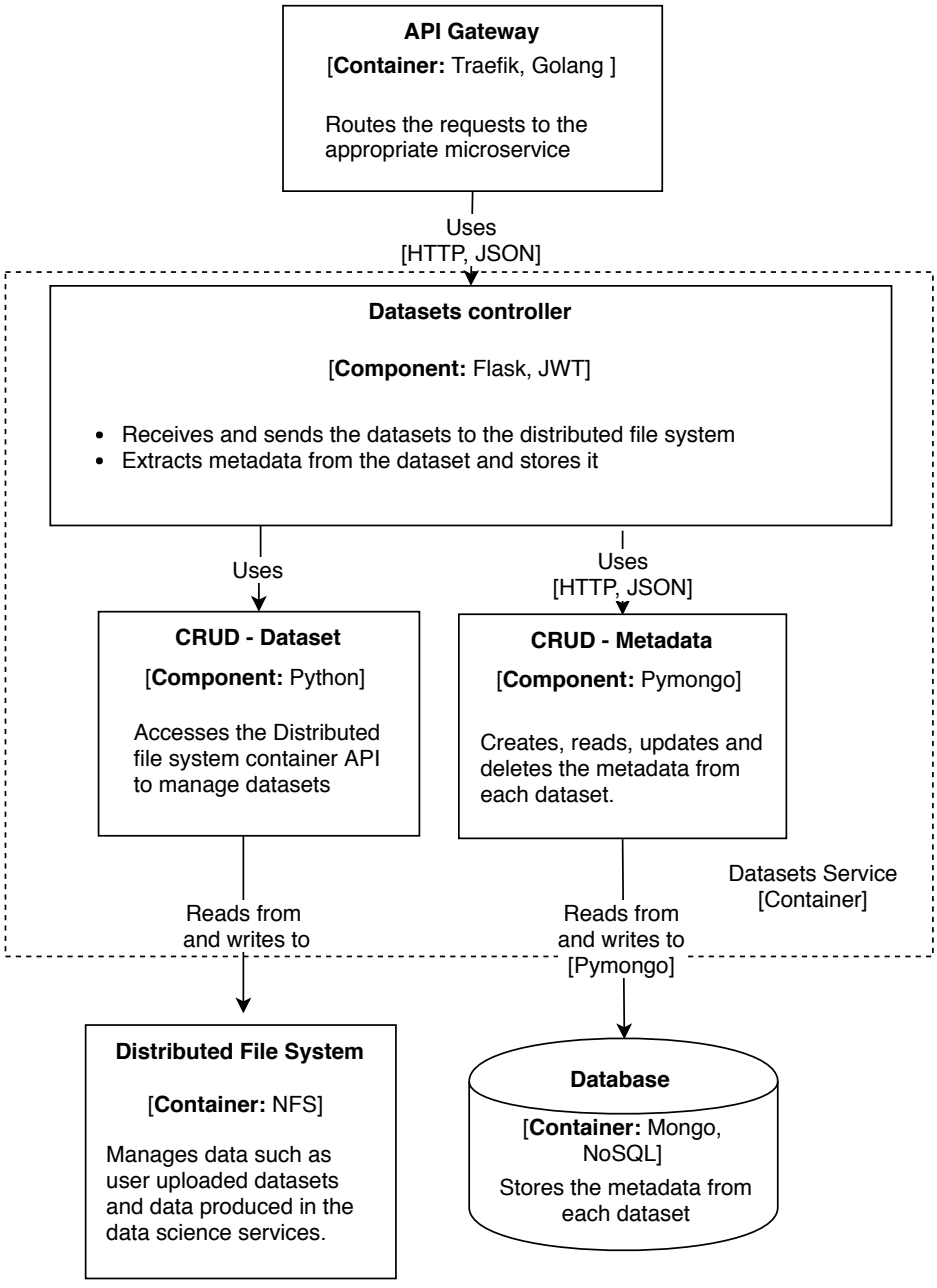

Figure 5.4: Datasets service: level 3 view.

5.1.7 Tasks Service

The main goal of this service is to detail every task that can be inserted in a workflow. This specification is sent to the users' web browser and contains: name of the task, unique identifier, parameters and its type. Each task belongs to a type (pre-processing, feature selection, model creation, etc...) and each type has a name and description, and also has the list of the other types of tasks that can be inserted next in a workflow. This specification allows us to introduce a order for the tasks that guide the user during the workflow building process.

The six types and the tasks that can be used in the workflows are the following:

- **Dataset input:** This type of tasks allows the user to set the source for the data to be used in the workflow. It has only one task associated at the moment. This task allows the user to define the URI of the dataset to be used in the workflow. It also allows the user to remove some attributes from the dataset and to specify which attribute represents the class. The output of this task is the URI to the dataset used in the workflow. In the future, we plan to add a task of this type that allows the specification of data sources from external services to be the dataset input.
- **Validation procedure:** Tasks from this type specify a process to use in the construction of the subsequent tasks that compose the workflow.
  - **Use entire data**: Specifies that the tasks being created in the workflow should use all data. This task does not present any output to the user.
  - **Train-test**: Specifies that the tasks being created in the workflow will be part of a hold out procedure. In this task, the user can specify the proportions to be used in the train and test sets. He can also specify if the data must be stratified or randomised and what seed should be used as the random number generator for this operation. This task will output the training and test sets produced in the workflow.
  - **Train-validation-test**: Specifies that the tasks being created in the workflow will be part of a train-validation-test procedure. In this task the user can specify the proportions to be used in the training, validation and test sets. He can also specify if the data must be stratified or randomised and what seed should be used as the random number generator for this operation. This task will output the train, validation and test sets produced in the workflow.
  - **Cross-validation**: Specifies that the tasks being created in the workflow will be part of a cross-validation or nested cross-validation procedure. In this task the user can specify the number of folds to be used, the number of repetitions and if it should be used the nested or the normal method. It is also possible to specify if the data must be stratified or randomised and what seed should be used as the random number generator for this operation. This task will output the train and test sets produced in the workflow for each fold.
- **Preprocessing:** contains tasks that apply transformations to attribute values present in the dataset.
  - **Feature scaling**: Allows the users to scale the dataset being processed in the workflow using the min-max method. The operations are only applied to the numerical attributes present in the dataset and never to the class. The outputs of this task are the datasets produced in the workflow during train and test, the

attributes used in the operation and an object to apply the scaling operation to new data.

- **One hot attribute encoding**: Allows the users to convert attribute values present in the dataset being processed in the workflow to a binary representation. Each attribute to encode will give rise to a new binary attribute that will be 1 in the rows where the value was previously present and 0 in the other rows. Here, the user can specify the attributes to be included or excluded from the operation. This task will output the names of the attributes included in the transformed dataset and the transformed datasets used during train and test of the best model.
- **Feature standardization**: Allows the user to apply z-score normalisation in the dataset being processed in the workflow. It applies z-score normalisation to all numerical attributes, except to the class attribute. The outputs of this task are the datasets produced in the workflow during train and test, the attributes used in the operation and an object to apply the normalisation operation to new data.

- **Feature selection:** contains tasks to assess the relevance of attributes for selection.
  - **Relieff**: Allows the user to assess which are the most relevant features in the dataset being processed in the workflow. In this task, the Relieff algorithm is employed with inputs useful for this algorithm, that are defined by the user. The user can also specify if he wants a threshold to be applied after running the algorithm to discard the attributes that score below such threshold. Besides these inputs, it is also possible to specify several numbers of attributes to select, separated by commas. When this parameter is set with several number of attributes, the model being built in the workflow will be created with the different numbers of best features according to Relieff to detect what is the best combination of features that must be included in the final model. The outputs for this task is the ranking produced with the algorithm using the training dataset, and the number of best features selected in the best model. When using cross-validation, the average and standard deviation of the scores produced in the different folds used to assess the best model are displayed.
  - **Info gain**: This task has the same job as the previous one. However, instead of receiving inputs useful for the application of the Relieff algorithm, it receives inputs useful to calculate the information gain of the features included in the dataset being processed. This task will display the outputs with the same logic presented for the previous task, however in this case the scores will be the information gain calculated for each attribute.

- **Model creation:** contains tasks for the creation of models using different algorithms.
  - **Support Vector Machine (SVM)**: This task produces and tests an SVM model using the training and test data being processed in the workflow. It receives inputs necessary during the creation of the model such as the regularisation parameter C and the kernel configurations. It enables the user to set different regularisation parameters and different kernel configurations to produce different models from the different configurations, from which the best model is presented to the user in the end. The outputs for this task are an object with the produced model, the parameters used to build the best model and the predicted and expected values produced after testing the model using the test data.

  - **Gaussian naive bayes**: This task produces and tests a gaussian naive bayes model using the training and test data being processed in the workflow. It does not receive any input parameters. The outputs for this task are an object with the produced model and the predicted and expected values produced after testing the model using the test data.
  - **Multinomial naive bayes**: This task produces and tests a multinomial naive bayes model using the training and test data being processed in the workflow. It does not receive any input parameters. The outputs for this task are an object with the produced model and the predicted and expected values produced after testing the model using the test data.
  - **Nearest neighbors**: This task produces and tests a k-nearest-neighbors model using the training and test data being processed in the workflow. It receives the parameter k that specifies the number of nearest neighbors to be used to train the model. The outputs for this task are an object with the produced model, the parameters used to build the best model and the predicted and expected values produced after testing the model using the test data.
  - **Decision tree**: This task produces and tests a decision tree model that uses the CART algorithm and using the training and test data being processed in the workflow. It does not receive any input parameters. The outputs for this task are an object with the produced model and the predicted and expected values produced after testing the model using the test data. This task also outputs an SVG image with the tree representation.

- **Model Evaluation:** contains tasks that specify the metrics to use for performance evaluation.

  - **Classifier model eval**: Allows the user to specify the classification performance metrics to be presented according to the best model produced in the workflow. It also lets the user specify what metric should be used as the decider of the best model configuration. This task outputs the classification performance metrics associated with the best produced model and also the confusion matrix associated with the tests conducted with the model.

An alternative to storing this information and accessing it trough the Tasks service, would be to hard code the information on the GUI. Its drawback would be that it would prevent external systems from accessing it and each time a new task was added to the system the GUI would need to be changed. Instead of doing that, this dynamic solution increases the systems capability to adapt to new changes and allows us to add new data science tasks at run time.

Figure 5.5 represents the main components of this service.

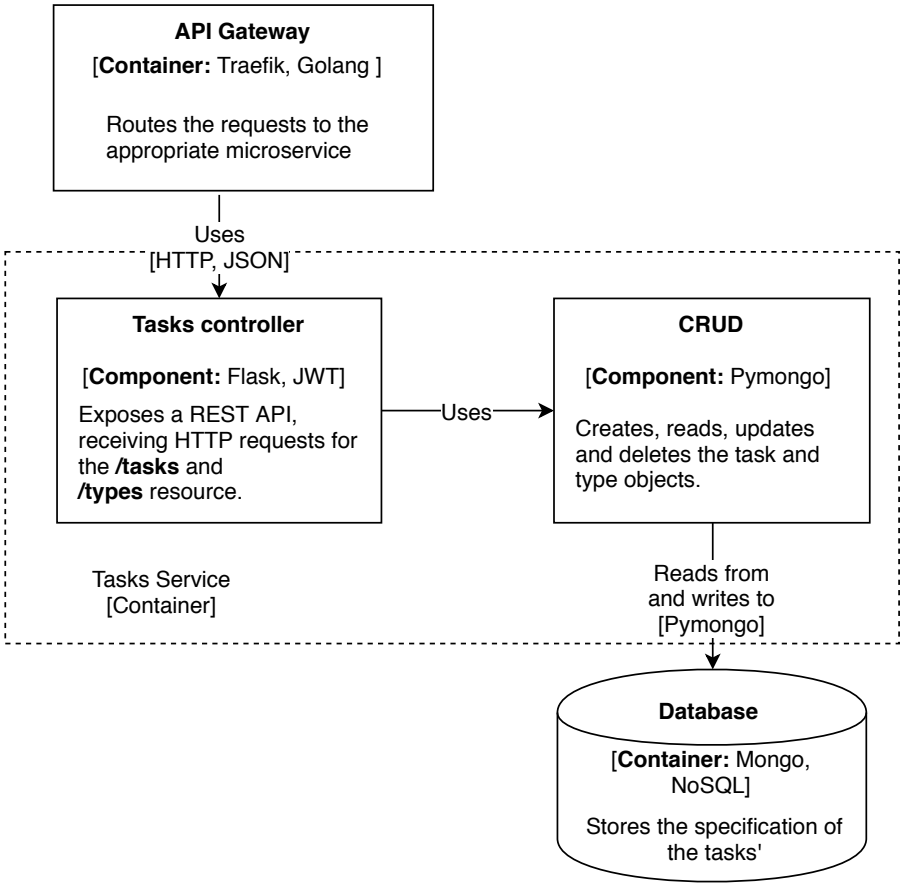

Figure 5.5: Tasks service: level 3 view.

5.1.8 Workflows Service

Almost certain to be the most complex microservice that was implemented, the workflows service is responsible for managing the workflows of the users and interfacing with the orchestration service to manage its execution. When the user does not specify that he wants to execute the workflow, this service basically just updates, deletes or saves the workflow objects in the database. However, when the user posts a workflow with the parameter start=true, besides saving the workflow, it will parse the workflow to create a new lower level and parallelised version of the workflow and then send it to the orchestration service to execute. During the parsing phase the translator also verifies if the order of the tasks is valid and catch most invalid values that could be inserted in the tasks' parameters.

We use sequential workflows to not force the notion of parallelization on the user, but many tasks gain from it. That is one of the reasons why we need to translate the high level workflows into lower level ones (system workflows).

The translation to the system workflows accomplishes two goals: The first goal is the paralellization of certain tasks as depicted in Figure 5.6.

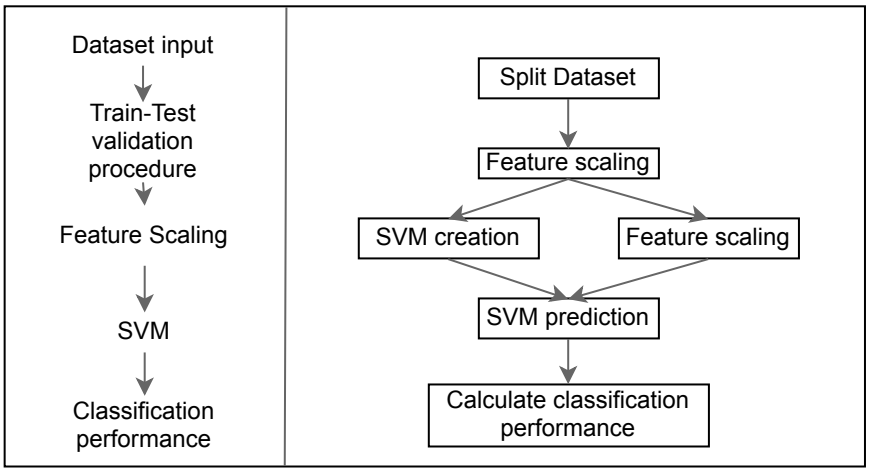

Figure 5.6: Example of a data mining workflow translation.

In this example we apply feature scaling only to the train portion of the dataset and then, during the training of the SVM model, we scale the features that will be used later for testing the model. The rules for the translation of the workflows are a work that will be continuously improved during the system lifetime and we are sure to be on the right path to satisfy even the most demanding performance requirements.

The second goal of the translator is to do parameter optimisation, more specifically grid search. There are certain parameters used for the creation of a model that are used to tune it; depending on the dataset used it could have different values. To give an example, when building an SVM model, the C parameter number is used to tune its generalisation capabilities; a number that does not fit well with a particular dataset will decrease the performance of a model. Usually, to find a good C value, a data scientist will need to create many models with different C values and check their performance to find a good number. What grid search does is precisely this, given a list of parameters it will test all of them in different combinations to find the ones that produce the best models.

The user could run a workflow with one C value, wait for its completion, check the performance and then run again with another value. He could keep doing this until he is satisfied with a good value. However, this would be a cumbersome task and would take much more time than it would have if the models were created with the different parameters at the same time. During the workflow construction process, the user can introduce many values for each parameter of a model creation task and the system will make sure to test all of them and show the ones that were used to create the model with the best performance.

Since the workflow is separated in tasks, the execution is also separated in tasks. Each task has the same pre-defined inputs and outputs regardless of the workflow that it is in. However, the tasks present in the workflows change constantly depending on the goals of the user. To execute these ever changing workflows we need to define a new system workflow each time we try to execute a high-level workflow. This definition is done using the Python library Condu, developed by the candidate. A brief description of its role for this service specifically is to facilitate the communication with the orchestration service and to specify the system workflows that will later be executed. In section **5.2** we will describe **Condu** to a greater detail.

Figure 5.7 illustrates the main components present in the workflows service.

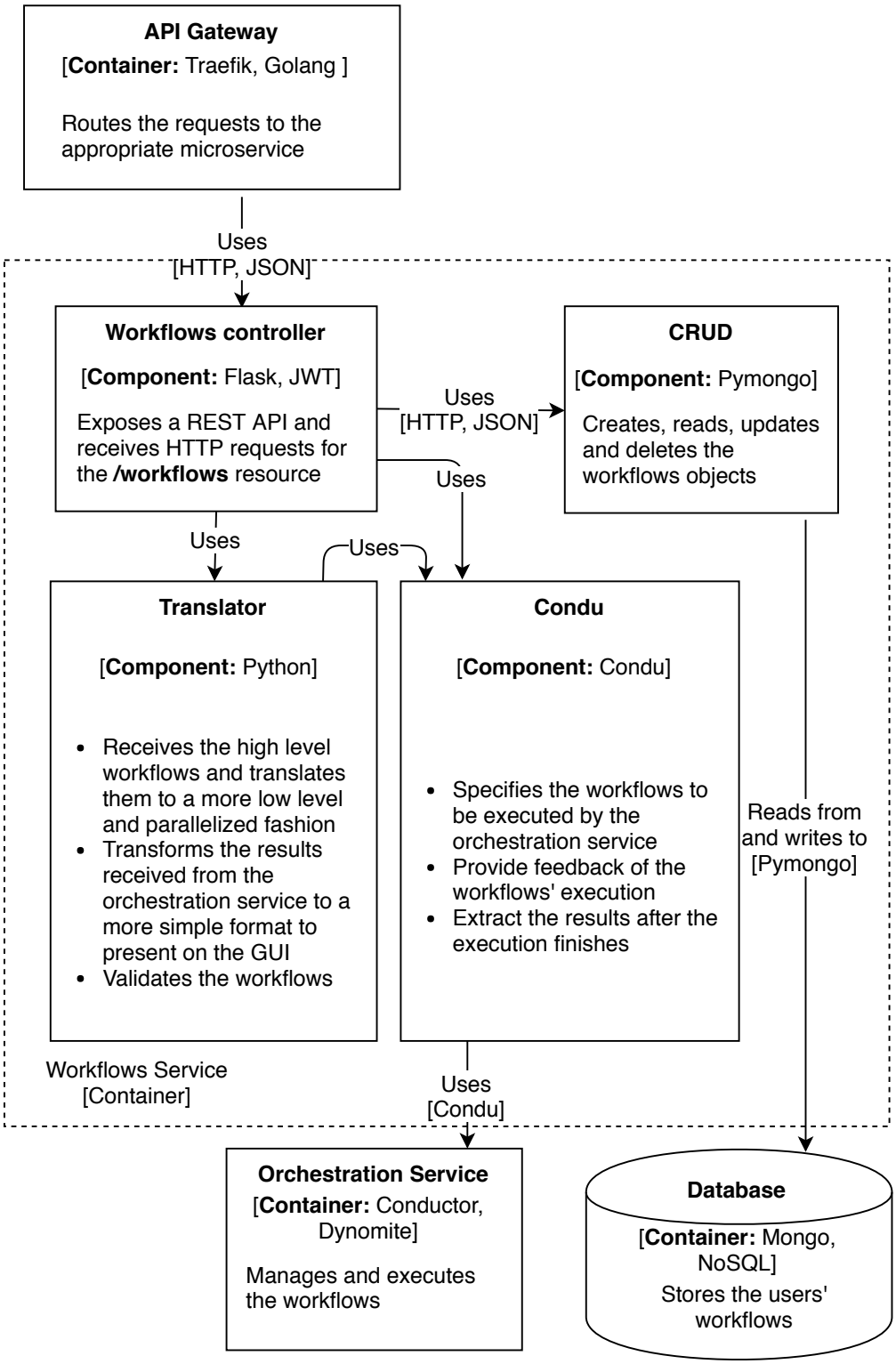

Figure 5.7: Workflows service: level 3 view.

5.1.9 Orchestration Service

In contrast with the other microservices, the orchestration service did not require any kind of implementation to develop; it is composed entirely by **Conductor**. This technology showed to be a great fit for the DS4NP platform. It allows the creation of dynamic

workflows at run time and their execution by scheduling the tasks in a queue and the workers (data science services) polling the queue to execute them.

Section 2.2.3 of the state of the art explains the motives behind the adoption of this technology. The details of how the orchestration is performed are explained in the orchestration section (5.2).

Figure 5.8 illustrates the main components present in the Orchestration Service.

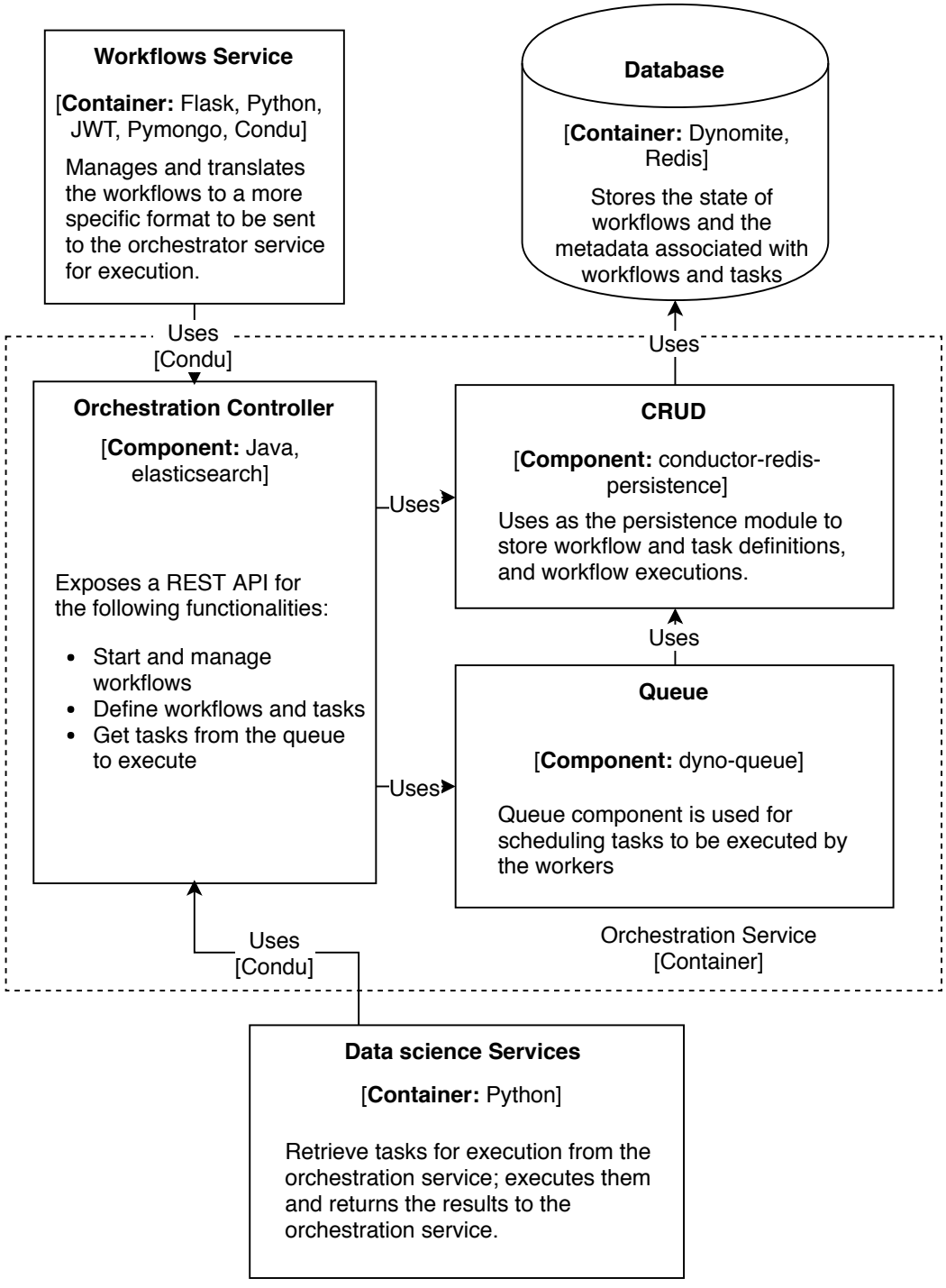

Figure 5.8: Orchestration service, level 3 view.

5.1.10 Data Science Services

This service is what is commonly called a consumer/worker in producer-consumer systems. In this case the orchestration service produces the tasks by putting them in a queue and the consumers (data science services) will retrieve the tasks to execute. The process of polling the orchestration service for tasks, trigger a function to execute when it receives a task and send the results and errors is performed using the **Condu** library. The execution of a task itself, is done recurring to common libraries such as **scikit-learn** or **pandas**. This reduces the amount of code needed to create the services and facilitates our job, since we do not need to implement the Machine Learning (ML) algorithms ourselves

Even tough we recur to libraries such as **scikit-learn** to implement the needed functionalies, these services are still complex due to the parameterization and configuration that they require.

Figure 5.9 illustrates the main components present in the data science services.

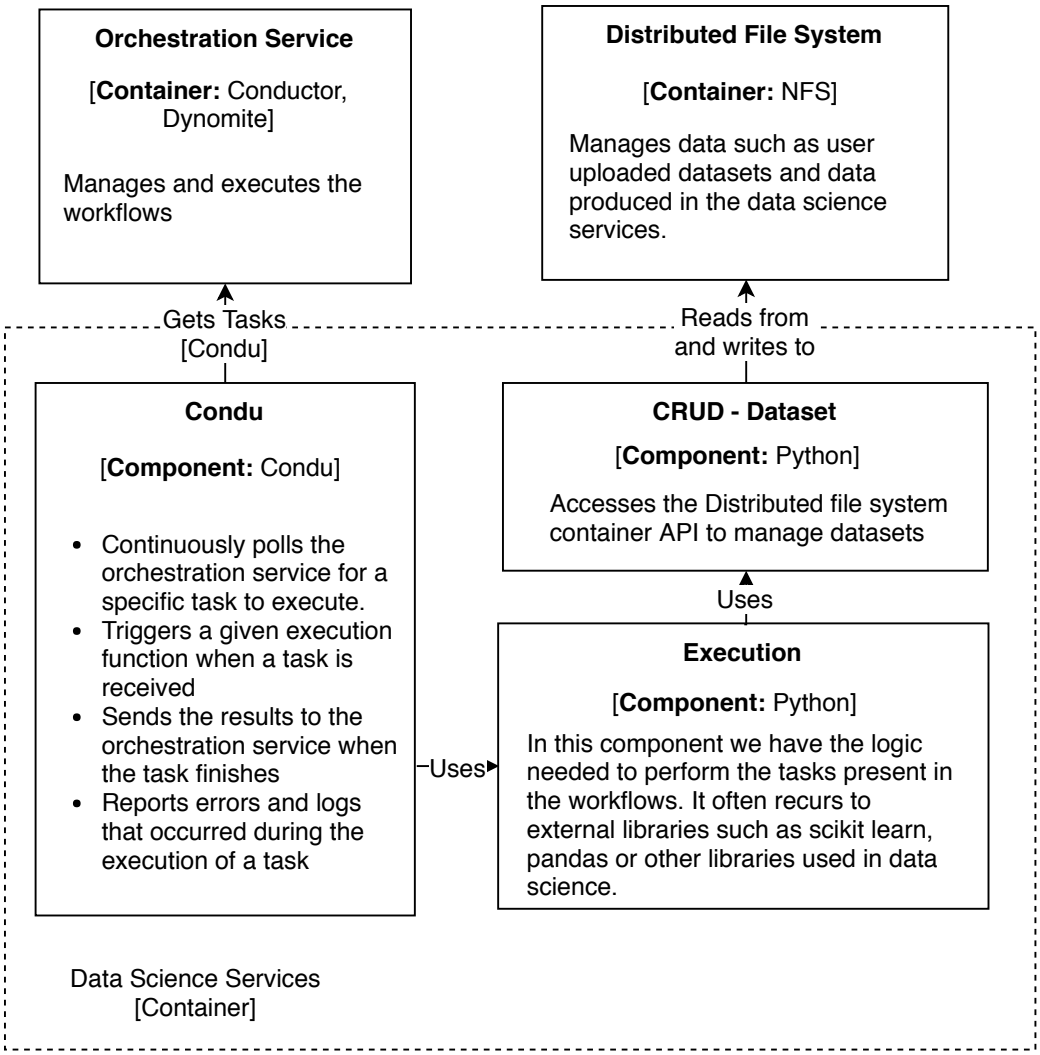

Figure 5.9: Data science services, level 3 view.

5.1.11 Distributed File System

While technically not a microservice, the DFS (distributed file system) is responsible for storing the datasets and allowing their deletion and updates. At the moment it uses the **NFS** protocol **RFC7530** [23], but this is just a temporary solution for the prototype. Even tough in our benchmarks and tests we use the NFS, it does not inherently support big data nor does it tolerate data corruption or allow data replication. We are studying Apache Hadoop [3] to become part of the solution. We are still investigating a permanent solution; we speculate that by using Hadoop, the DFS will allow the suport of big data and tolerate data failures. This change does not bring any significant impact to the architecture and will allow us to decouple the DFS with the other microservices.

5.2 Orchestration

As previously described in section 5.1.9 the orchestration of the system workflows is performed using **Netflix Conductor** (Orchestration service). All the interactions with Conductor can be made using HyperText Transfer Protocol (HTTP) connections to its well defined REST API. The workers can poll it to execute tasks and to update the results of those tasks. This interface follows the same modularity standards as the other microservices and it is not tightly coupled with any other.

Having that said, we developed the Python library and framework **Condu** to facilitate the interaction with the orchestration service. Any microservices that would be added in the future are not restricted to use Condu. Nevertheless, if that microservice is implemented in Python the use of Condu will be of great value and decrease the difficulty of implementation. Since **Condu** is an tool to utilise the functionalities of the orchestrator, we hope that by explaining Condu's functionalities the orchestration will be inherently explained as well.

Condu has two main roles:

- Task and workflow definitions
- Worker and workflow execution

In the next sub-sections we will explain each role individualy.

5.2.1 Task and Workflow Definitions

Conductor needs to have the tasks and the workflow process defined to enable its execution. The definition of a task is done to configure which behaviour a task should have when there is a failure. In other words it allows us to respond to the following questions:

- Should a task be rescheduled when it fails ? How many times?
- How much time should be waited before rescheduling a task ?
- Does the task have a timeout ? How long is the timeout ?

Using the object **TaskDef** imported from Condu allows us to specify what should happen in these scenarios.

Figure (5.10) illustrates how this is done.

```
from condu import Condu, TaskDef

task=TaskDef('task_name')

# String  | An unique name used to identify the task.
# This is the only attribute that is required, all
# others are optional.
task.name='unique_name'

# Integer | Number of times the task is rescheduled
# after being marked as failure.
task.retryCount = 0

# Integer | Number of seconds waited to reschedule a task after failing.
task.retryDelaySeconds = 0

# String  | Can take the following values:
#   'FIXED' : Reschedules the task after retryDelaySeconds
#   'EXPONENTIAL_BACKOFF' : Reschedules after retryDelaySeconds  * attempNumber
task.retryLogic='FIXED'

# Integer | Time in milliseconds, after which the task is marked
# as TIMED_OUT if not completed after transiting to IN_PROGRESS status
task.timeoutSeconds = 0

# String  | Task's timeout policy
#   'RETRY' : Reschedules the task
#   'TIME_OUT_WF' : Workflow is marked as TIMED_OUT and terminated
#   'ALERT_ONLY' : Registers a counter (task_timeout)
task.timeoutPolicy=None

# Integer | Number of seconds to reschedule a task if it is
# not updated with a status. Useful when the worker polls for the task
# but fails to complete due to unexpected errors or a network failure
task.responseTimeoutSeconds = None

# Sends the TaskDef object to the Orchestration Service
Condu(orchestration_endpoint).create_task(task)
```

Figure 5.10: Example of a task definition

After defining the tasks we can now start creating workflows. To create a workflow we need to use the **WorkflowDef** object. Its job is to specify the order of the tasks and the inputs of the workflow. The order is defined in a list of WorkflowTaskDef objects. Each object references a task and is used to do the wiring of inputs by specifying the sources of each input, i.e., we connect the outputs of one task to the inputs of another.

Figure 5.11 is a simple example of how to define a workflow.

```
from condu import WorkflowDef, WorkflowTaskDef

workflow = WorkflowDef('simple_workflow_name')
workflow.inputParameters = ['foo', 'bar']

# Specify the tasks that will be inserted
task1=WorkflowTaskDef('simple_task1')
task1.intputParameters={'foo':'bar', 'some_number':420}
task2=WorkflowTaskDef('simple_task2')
# If task1 has 'result1' has an ouput, the following line wires
# that output to task2 input.
task2.inputParameters={'input1':task1.get_path('result1')}
sample_workflow.tasks[task1,task2]
Condu(orchestration_endpoint).create_workflow(workflow)
```

Figure 5.11: Example of a workflow definition

The lines of code required to define workflows increase depending on their complexity and number off tasks. The workflow provided above is sequential and very simple. There are two kinds of tasks: tasks that are executed by the workers ; and logic tasks that are executed by the orchestrator. The tasks executed by the workers are represented by the WorkflowTaskDef object, the other types are:

- **ForkTaskDef**: used to schedule tasks to execute in parallel.
- **JoinTaskDef**: used to wait for the completion of one or more tasks spawned by fork tasks.
- **DecisionTaskDef**: is similar to a **switch** statement used in programming languages such as C or Java.
- **SubWorkflowTaskDef**: allows us to execute another workflow as if it was task.
- **DynamicTaskDef**: used to execute a task that is only known at run time.
- **DynamicForkTaskDef**: works the same way as **ForkTaskDef** except that the list of tasks to be forked is provided at runtime.

The dynamic tasks are particularly useful when the name or the number of tasks to be executed are not fixed and varies based on input. Aside from defining tasks and workflows, Condu also allows to delete, update and search for tasks or workflows previously defined.

5.2.2 Worker and Workflow Execution

To manage the workflows Condu has the following methods:

- **start_workflow**: starts the execution of a workflow. Its parameters are name and the inputs for the workflow.
- **get_workflow**: retrieves an object that represents the current state of the workflow execution. Its parameter is an id that identifies the execution.

- **get_task_from_workflow**: Similiar to **get_workflow** but it just extracts the state of a specific task. Useful when we want to know if a certain task was finished, and if so, extract its outputs.
- **terminate_workflow**: Terminates the execution of a workflow. Its parameter is the execution id.
- **pause_workflow**: Not used in the DS4NP platform but a workflow can be paused and resumed at a later date. Its parameter is an id that identifies the execution.
- **resume_workflow**: resumes the execution of a workflow. Its parameter is an id that identifies the execution.

Condu abstracts the hassle of polling for tasks, allowing the developer to focus on implementing the task itself. Figure 5.12 exemplifies a data science service (worker) that executes the feature scaling task in a dataset used for training.

```
from condu import Condu

def feature_scaling_train(task):
    # Retrieving the task inputs
    dataset_uri = task.inputData.get('dataset_uri')
    # The scaling is done here
    .
    .
    .
    # Inserting the results as the output
    task.outputData =
    {'dataset_uri': dataset_uri,
    'scaled_attributes': scaled_attributes,
    'scaler_uri': scaler_uri}
    task.status=COMPLETE

condu = Condu(orchestration_endpoint)
# Specifies the name of the task and function to execute
condu.put_task('feature_scaling_train', feature_scaling_train)
# Specifies number of parallel processes that will
# execute the tasks and the polling interval
condu.start_tasks(processes=1, polling_interval = 0.2)
```

Figure 5.12: Example of a worker using Condu

Every **exception** (execution errors) that might happen during the execution of a task that are not handled by the developer, is caught by **Condu** and reported to the orchestration service, marking the task as FAILED and storing the logs. This is done transparently and increases the performance of the platform by marking the task as FAILED early and not waiting for its timeout.

5.2.3 Retrospective

The use of **Condu** had a pivotal role in the execution and creation of workflows in the workflows and data science services. Netflix already had a Python and a Java client

library hosted on GitHub to communicate with Conductor. After an extensive analysis of the Python client and reading all its code the main reasons for not adopting it were:

- It was written in Python 2.7. This version of the Python language will stop being officially maintained at the start of 2020. Making it a temporary solution and not worth the investment.
- Netflix says this client is still in development but we did not see a code commit in over 5 months.
- We detected bugs in a few functions. A code commit that fixed these bugs was issued on GitHub on the 1st of March, 2018, but was only approved on the 10th of August, 2018. These changes will be included in the next major version release but it goes to show that the development of the client does not have a lot of focus on the Netflix side.
- It only supported workflow management (start, terminate, get workflow status etc...) and the execution of tasks by the workers. In contrast, the definition of tasks and workflows was not supported.
- The code necessary to use it seemed a lot verbose for a Python library.

The first step to create Condu was to try and find the parts of code on the Netflix's client that could benefit the creation of a new Python 3 version. The code was refactored and translated to Python 3.6. Next, we added the code for the definition of tasks and workflows, and added functions to facilitate the extraction of results from running/finished workflows. As the final step, we improved the performance of the execution of tasks by using processes instead of threads. This change increases the performance when a worker is executing multiple tasks due to the fact that, in Python, because of the Global Interpreter Lock (GIL), threads are not truly parallelized, only concurrent.

Condu has 734 lines of code without accounting for the unit tests. 302 lines of that is code that originates from the code that was refactored from Netflix. Regarding the Java library it had the same functionalities as the one they provide in Python and was used to execute the **Relieff** task.

5.3 Graphical User Interface

The GUI was built using Javascript and HTML with the purpose of running in the user's browser in order to allow any personal computer with internet access the use of the DS4NP platform. The design had the objective of creating a minimalist and simple application where the user can produce the most value with the lowest amount of effort. **ReactJS** was the main framework used to create the interface. It runs entirely on the browser and communicates using HTTP requests to the system. There are many alternatives to ReactJS such as AngularJS and VueJs. They are very similar but we chose ReactJS because we were more familiar with it.

The GUI is divided to two key areas: the sidedar, and the staging area where workflows are built. As we can observe in Figure **5.13**, the darker area on the left is a sidebar; it allows the creation and deletion of workflows, and uploading, downloading and deleting datasets. Also, when a workflow is present in the staging area, because the user created or opened one, it is also possible to save a workflow or start/stop its execution.

The staging area is seen on the right and allows the user to build workflows by inserting the tasks described in section 5.1.7.

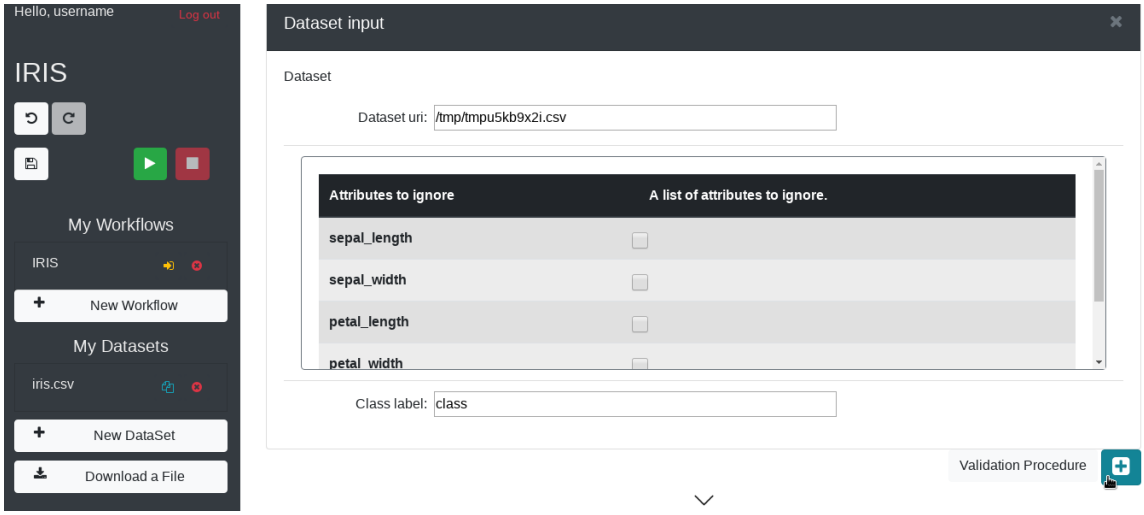

Figure 5.13: GUI example

Every workflow is clearly divided to separate blocks (tasks). The starting task is allways of the dataset input type and from then on the user can choose to insert tasks bellow, delete or change the default parameters of the tasks in the workflow.

Each time the user opens the GUI, it retrieves the description of the tasks and types from the tasks service. Doing this allowed us to build a dynamic interface where the addition of new tasks or changes to them does not imply changes to the interface.

When inserting a task, the user is shown the types of tasks that can be added next (Figure 5.14). This means that when the user clicks the 'plus' button to insert a task, depending on the current state of the workflow, he can only see the tasks that can follow and is not cluttered with all tasks at once. By doing this, the tasks are chained together, guiding the user during the construction process.

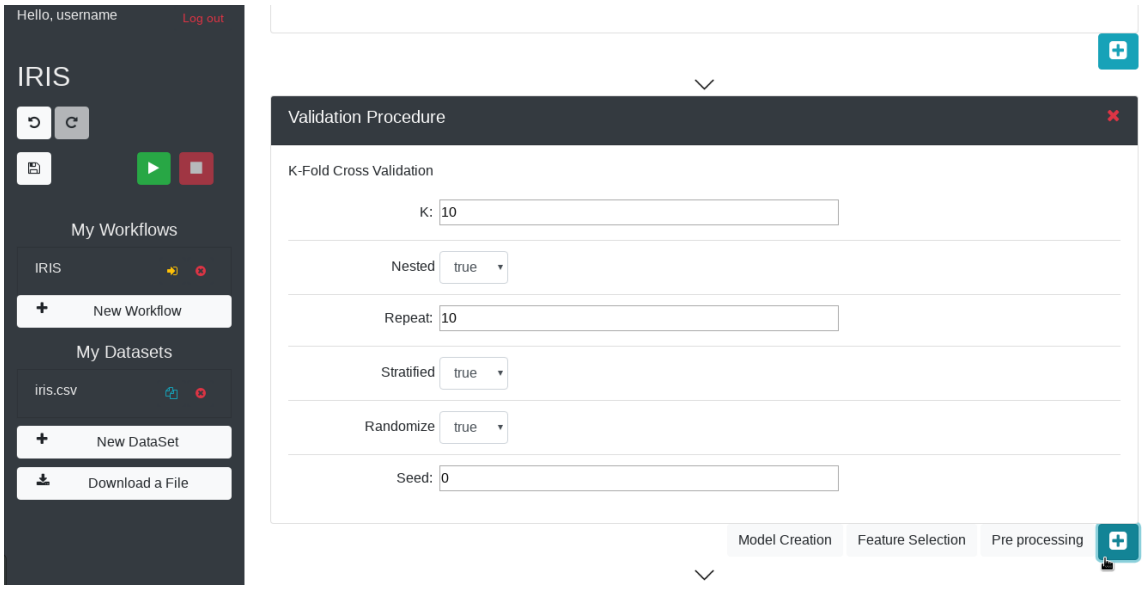

Figure 5.14: Inserting a task in a workflow

We found inspiration in GUIs that use the sidebar as some kind of toolbox such as AzureML, Clowdflows and Orange. We also made sure that the staging area does not distract the user from the building process; hence, functionalities that do not pertain the building process are shown in the sidebar instead.

This page is intentionally left blank.

Chapter 6

Testing

By formally testing each aspect of the system we can evaluate it and find whether it satisfies the specified requirements. In this chapter we will describe the tests and strategies used for this analysis. Starting with unit testing, followed by usability testing, and finishing with benchmarks to test the performance of the system.

6.1 Unit Testing

To test the functionalities of each component of the system, we employed the testing strategy know as **black-box**. By using this strategy, the structure of each component was not taken into consideration. We analysed the outputs that were generated in response to selected inputs and compared them to the expected results; if the outputs are the expected ones, the test is considered a success.

According to the designed architecture, the system is divided to modular and independent services. This allowed us to test each service separately as independent programs. The tests were written in Python and the **requests** library was used to make the HTTP requests to the services. In order to run all the tests and compare the ouputs to verify if the tests passed or failed, the framework **pytest** was used.

Figure 6.1 shows us running the tests written for **Condu**.

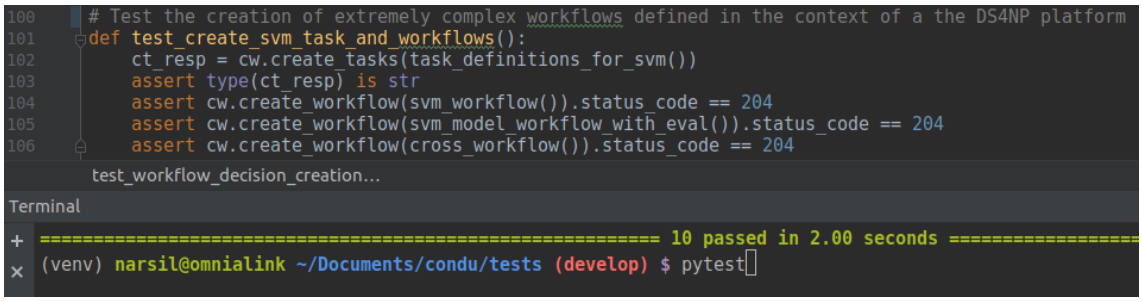

Table 6.1: Unit testing the Condu library

The tests revolve around each service resources. For instance the workflows service has the '/workflows' resource that is provided and the tests focused on testing the methods for it (GET, POST and DELETE). The tests for the other services will follow the same pattern. They are still not formally tested and are still being developed. At the moment, due to

being the most critical, the services that were formally tested are the workflows service and orchestration service. The tests performed are very simplistic and do not test the services well enough. Further tests with higher complexity and involving many services at the same time need to be created to allow the system to transition from prototype to a system that is production ready.

6.2 Usability Testing

In this section we will describe the process and the results of the usability tests. In experimental setup sub-section we describe the structure and the two groups of participants. Followed by the description of the exercises that they performed. Finishing with the results from the tests.

6.2.1 Experimental Setup

The usability tests provided a crucial role in evaluating the prototype and validating the paradigm of visual programming using sequential tasks. The tests consisted in having the participants execute a few exercises using the interface and getting their feedback. This feedback was then used to evaluate the participants' experience, the usability of the interface, and the value that was provided to them, hence validating this concept of visual programming applied to data science.

There are two distinct populations of participants:

- **Type A:** Participants with no experience at all and no knowledge in data mining, composed of a group of researchers, four with a masters degree in ecology and three with a doctoral degree in biology (7 participants).
- **Type B:** Participants that were knowledgeable about data mining but were not programmers. This group was composed of students who were enrolled in a master's degree in biochemistry and were undertaking a course in data mining (11 participants).

The process was separated in different steps: The first step started with a quick overview of the platform and its functionalities, which took less than 3 minutes. After this introduction and answering any questions the participants had, we gave them a paper with a problem and a list of exercises for them to perform in order to solve that problem. The exercises fundamentally consisted in using the data science tasks mentioned in section 5.1.7. If the participants successfully finished the exercises they would have solved the problem. This challenge was estimated to take about 20 minutes. The last step was a questionnaire that the participants had to fill about their experience, and their thoughts on the relevance of this platform. The questions were written in Portuguese but were translated for this thesis.

6.2.2 The iris flower dataset problem

To keep the tests brief and not overly complicated we decided to introduce one of the common problems new data scientists learn during their training: the iris flower dataset. This data was collected by Edgar Anderson to quantify the morphological variation in

iris flowers of three related species [2]. It contains a total of three species of iris: Iris Setosa, Iris Versicolour, Iris Virginica; and consists in the dimensions of its petals and sepals (centimeters). Table 6.2 depicts what the dataset looks like.

sepal_length	sepal_width	petal_length	petal_width	class
5.1	3.5	1.4	0.2	Iris-setosa
4.9	3	1.4	0.2	Iris-setosa
4.7	3.2	1.3	0.2	Iris-setosa
4.6	3.1	1.5	0.2	Iris-setosa
5	3.6	1.4	0.2	Iris-setosa

Table 6.2: Iris dataset

Based on the measurements, the participants would then create a model that could predict the species of iris. The test was separated in 5 exercises:

1. The first exercise consisted in scaling the attributes of the dataset between 0 and 1.

2. Exercise two required the participant to split the dataset to training and test sets (60/40%). The one for training would later be used to train an SVM model and the one for testing to see the accuracy and f-measure metrics.

3. Exercise three was similar to number two, however including a feature scaling operation before the model creation. This was conducted to verify whether the participant was aware that tasks could be created and removed in the middle of a workflow previously created.

4. In exercise four the participant was asked to add the Relieff algorithm to the workflow in order to see what attributes would have the most predictive capabilities.

5. Exercise number five used the best two attributes discovered in the previous exercise and added the validation procedure called K-fold cross validation, hence completing the assignment and creating a model with high accuracy.

The exercises were simple and intertwined making the participants have a feeling of progress during their execution.

6.2.3 Results

Questionnaire

The questionnaire allowed us to know how much the participants liked the interface, their experience using the tool and whether they found it useful. Each statement could be answered as: totally disagree, disagree, indecisive, agree and totally agree. In order to analyse the average response and the standard deviation we converted the answers to numbers, where number 1 translates to “totally disagree” and 5 to “totally agree”.

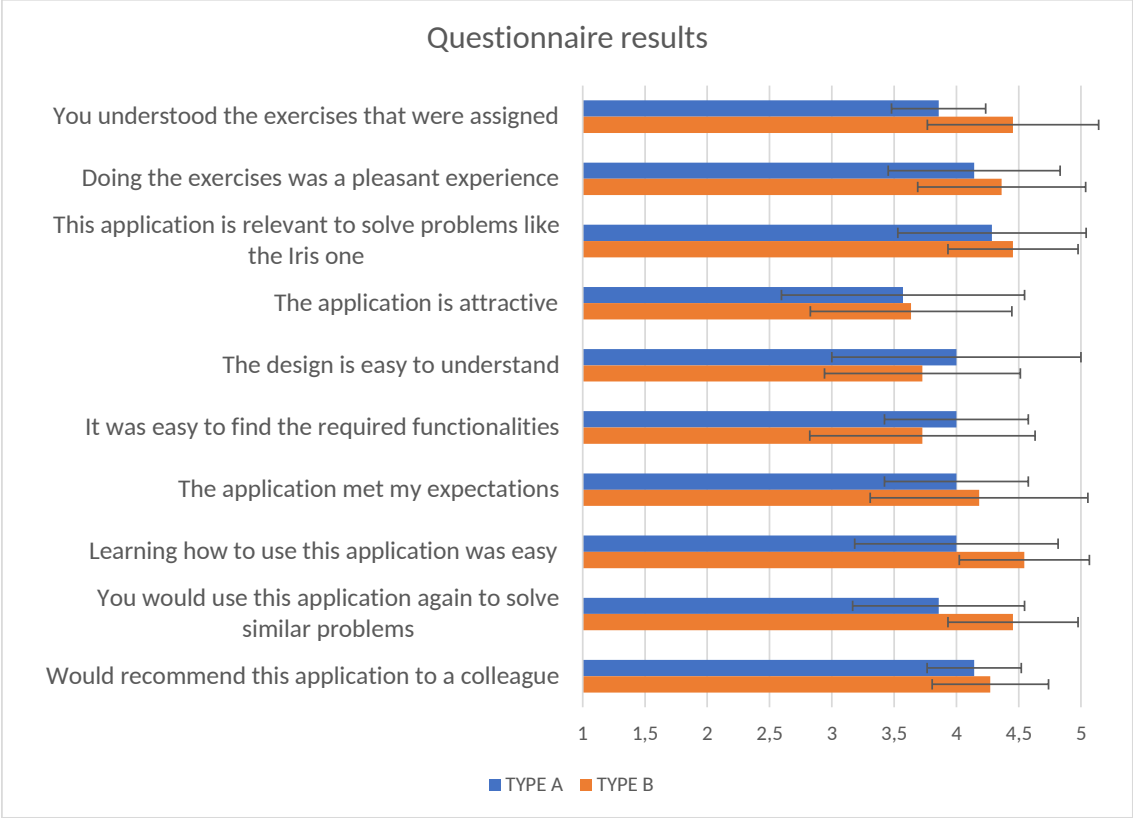

Figure 6.1: Average and standard deviation of the participants' responses

As seen in Figure 6.1 the values are all above average. The most satisfactory results where that they found the interface easy to use, they would recommend it to colleagues and that they would use it again to solve related problems. The attractiveness of the interface, even though it was very positive, scored lower than the other metrics; there was no surprise here since this is a prototype and that part was not a priority.

The overall results acquired from the participants with no experience (type A) are lower than the ones with experience (type B). This was expected and showed that the participants with no experience (type A) had an higher difficulty using the interface. Surprisingly, they found easier to find the required functionalities and the design simpler to understand.

To assess whether the differences in the answers among the two populations were statistically significant, we performed unpaired statistically significant tests. Before that, both distributions were tested for Gaussianity using the Kolmogorov-Smirnov test. In case both distributions were Gaussian, an unpaired T-test was conducted; otherwise, a Wilcoxon rank sum test was performed.

Hence, for each of the ten questions, only the question "You understood the exercises that were assigned" showed statistical significant differences among the two populations (at $p < 0.05$). We hypothesise that it was easier for the type B subjects to understand the exercises because they had experience in data mining and knew about the Iris dataset since it is a very common problem to teach new data scientists.

Log analysis

With the objective of analysing the participants' behaviour as mentioned in the Logs service section (5.1.3), during the usability tests, we collected logs to be later analysed with the purpose of knowing if have done the exercises correctly and estimate the time took to complete each one.

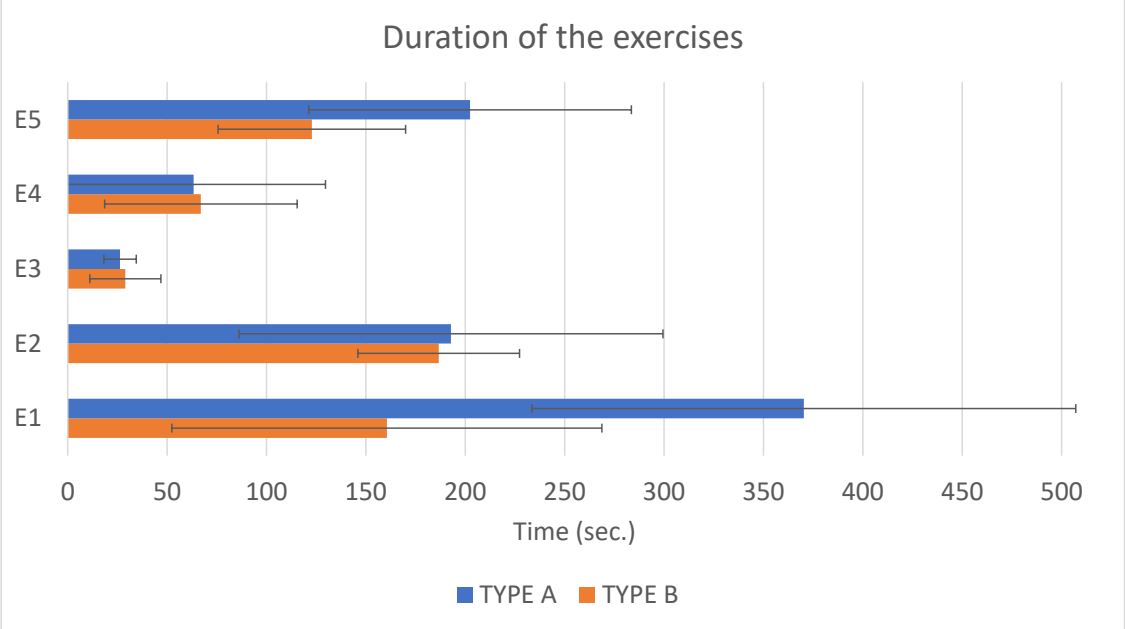

Figure 6.2: Average and standard deviation of the time taken to complete each exercise

As seen in Figure 6.2 the type A population had a much tougher time in the first exercise than the type B. Once again this result originates from the fact that they had never built a model. During the introduction to the experiment, both populations understood the objectives but the experience of type B participants proved to be a beneficial factor in reducing the time taken to complete the exercises. This finding is also corroborated with the fact that the type A population did not understand the exercises as much as type B. On average the type A population took **14.2** minutes to complete all the exercises and type B took **9.4** minutes, which was well bellow our maximum expected time of 20 minutes. It is important to notice that the participants often stopped doing the exercises to ask questions or even give suggestions right away. Even though we tried to not distract them to not affect the experiments, it was something that we could not prevent.

We were also able to determine the **effectiviness** of the exercises by measuring the percentage of exercises completed successfully using the following formula:

$$Effectiveness = \frac{Number\ of\ exercises\ completed\ successfully}{Total\ number\ of\ exercises\ undertaken} * 100\%$$

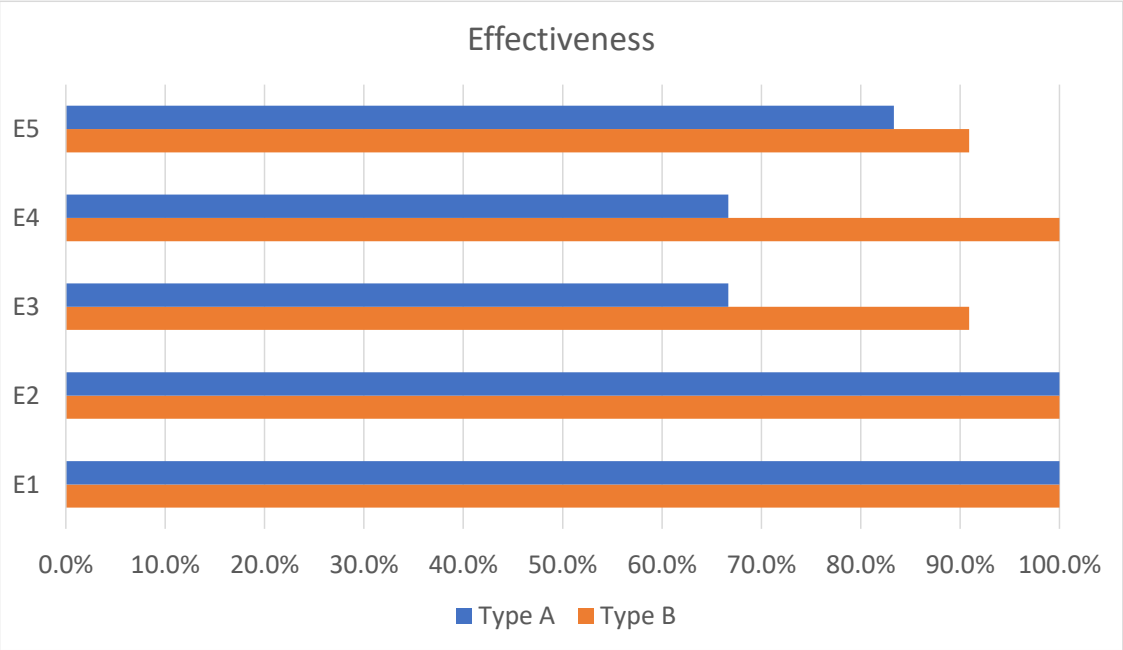

Figure 6.3: Average of the effectiveness for each exercise

As we keep observing with all the other results, the type A population performed worse than type B. The average effectiveness of the two groups combined was 91.78%. Specifically, the type A population averaged 83.30% and type B, 96.40%.

From this analysis we concluded that the effectiveness quality attribute specified in section 3.3.1 was in fact satisfied.

Feedback

Besides answering the questionnaire, the participants also had a place to write suggestions, critiques and things they liked better in the application. Many of them also wanted to express their suggestions verbally after doing the tests and to know more about the future of this system. In this section we will approach the key suggestions and critiques that we got from the test subjects.

Key **suggestions**:

- **It should be possible to see all the tasks that were added to the workflow at all times.**

  This suggestion was done because when the workflow is long enough, the user might not remember specifically what tasks or parameters were inserted in places that he can not see at the moment. The user needs to scroll up and down to see and edit the tasks. This is something that is worth investigating to minimise mistakes such as inserting the same task twice in different places of the workflow because the user forgot that it was already inserted somewhere else. This is something that might not be difficult to correct given the simplistic nature of the workflows.

- **In the dataset input, the option of selecting attributes to remove from the dataset should be replaced with attributes to select.**

At the moment the user is shown attributes to ignore/remove from the dataset. Some participants showed a preference to selecting attributes instead of removing them. This needs to be further investigated, because depending on the datasets one might be more beneficial than the other. A solution for this might be to have the option to use both of them.

Key **critiques**:

- Sometimes the participants did not know that a task belonged to a certain type, e.g feature scaling is a task of the preprocessing type but some participants when asked to use it did not intuitively know that it was of that type. This is a problem because the users need to first select the type of task they want to choose from, and then the task itself. This problem can be solved by finding an alternative for the way users add tasks to the workflow or having a place where the users can search for all available tasks and read more information about them (good documentation). At the moment we do not support that many different tasks and the participants found all tasks within a few seconds. However, if there were more tasks it would be difficult with the current interface.
- To use a dataset in any workflow the users must copy the dataset's uri that is shown in the sidebar and paste it to the dataset input task. Some participants did not find that intuitive. Another alternative should be taken to consideration.

Things participants **liked the most**:

- Simplicity, accessibility and design.
- Low learning curve and easiness to use.
- How fast it was to run an experiment and get the results.
- Intuitiveness.
- It does not require any installation and it can be used anywhere with internet access.
- The tasks were chained together guiding the process of constructing the workflow.
- Grid search.
- The outputs are direct and very informative.

This feedback also reinforced what was discovered during the questionnaire analysis and was very satisfactory. The critiques and suggestions made are things that will be improved as the development of the system continues. None of the critiques were about the concept we aim to prove. The things they liked the most were inline with the objectives we aimed to achieve when building the application, which was something that we believe helps validate our goals.

6.3 Benchmark

Basic computational performance tests were conducted to assess how the system will behave with the current architecture. We executed tests using two randomly generated

numerical datasets with a binary attribute class: Dataset 1 contains 10000 rows and 1001 columns (34.2 MB) and Dataset 2 has 20000 rows and 1001 columns (68.4 MB). Using each dataset we created a Naïve Bayes model, evaluated its classification performance using 10-fold cross validation and recorded the execution time. This process was repeated 10 times.

H2O was the platform used to benchmark against DS4NP because it was the only cloud solution that we could configure to have the same computational resources as our system. They were both deployed in equal clusters for the experiment and were composed by four virtual machines with 2 virtual CPUs and 7.5Gb of memory each.

The results of the average execution time can be seen in Figure 6.4.

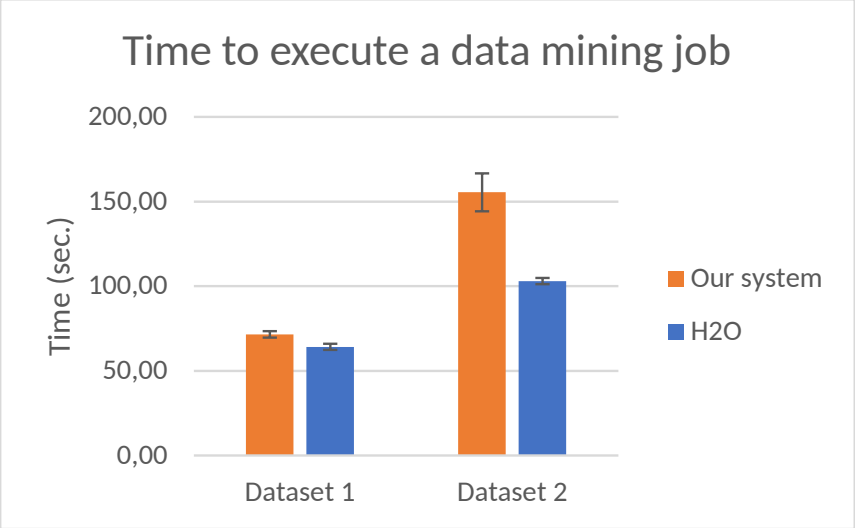

Figure 6.4: Tests performed with our system and H2O.

As we anticipated, our system was slower in this experiment. The main contributing factor for the different in performance might be because we are storing intermediate results in disk using a Network File System (NFS), while H2O stores them in memory. We will address this issue in the future. Even though H2O's solution gains in performance, it has the drawback that if one machine fails during execution, the whole process must be restarted. In our system, because of the orchestration solution that was adopted, this does not happen.

Chapter 7

Project Management

Thoughtful planning is essential to the success of a project. As such, in this chapter we present the stakeholders of the project, its development strategy and conclude with the work plan.

7.1 The stakeholders

The group of stakeholders for the Data Science for Non-Programmers (DS4NP) project is composed by Professors Dr. Filipe Araújo, Dr. Jorge Cardoso and Dr. Rui Pedro Paiva, which supervised and guided our work through the entire academic year. Besides the supervisors, a PhD student, Jaime Correia also gave guidance and participated in the fortnightly meetings of the project. The project development is taken by myself and another MSc Student, Artur Jorge de Carvalho Pedroso, that was specially focused on the **data science services** described in section 5.1.10.

7.2 Development methodology

At the start of this project we did not had a clear view of the path that could be taken. The first months were spent researching related works and cloud technologies. It was only closer to the end of the semester that we could start making decisions about goals and requirements for the DS4NP platform and designing its preliminary architecture. Since the first semester was spent mostly doing research, we felt that there was no need to adopt any software development methodology.

Having identified the general requirements for the project and technologies researched, we thought about a development methodology for the second semester. Considering that we are a small team of 2 MSc students, developing an experimental kind of software, for which the requirements and technologies can change very fast, we felt that an Agile development process would be a good fit. After searching some Agile methodologies, and finding that most of these methodologies were created for development teams having more than two elements, we identified the Kanban software development methodology as a potential candidate for the second semester. During the intermediate presentation for this thesis, it was suggested by the Juries that the method **timeboxing** would support much better the development of this project as opposed to Kanban. As a matter of fact, we were already doing something similar to timeboxing from the start, we just had not

read about it before. We had fortnightly meetings to show the progress of our work and to plan the work to do for the next meeting. After a thoughtful analysis we decided that this method should be improved with more demanding deliverables in the second semester.

We changed the time periods (time boxes) to a week. Everery week we had a scheduled meeting where we would present the deliverables that were planned in the former meeting. In contrast with the first semester, in the second one we started prioritising the requirements. The prioritisation was done according to the MoSCoW method described in the functional requirements (Section 3.2).

Every meeting had an individual presentation where I and Artur would present the work performed by each one, according with the goals planned. The presentations most of the times included a Powerpoint or/and had a demonstration of the functionalities that were added to the software system. After the presentation we would analyse the current state of the project, then we would plan the deliverables for the next meeting. This was an iteractive process and was very adequate for this project.

7.3 Work Plan

Even though our work revolved around time frames of a week, we also had long term goals to achieve.

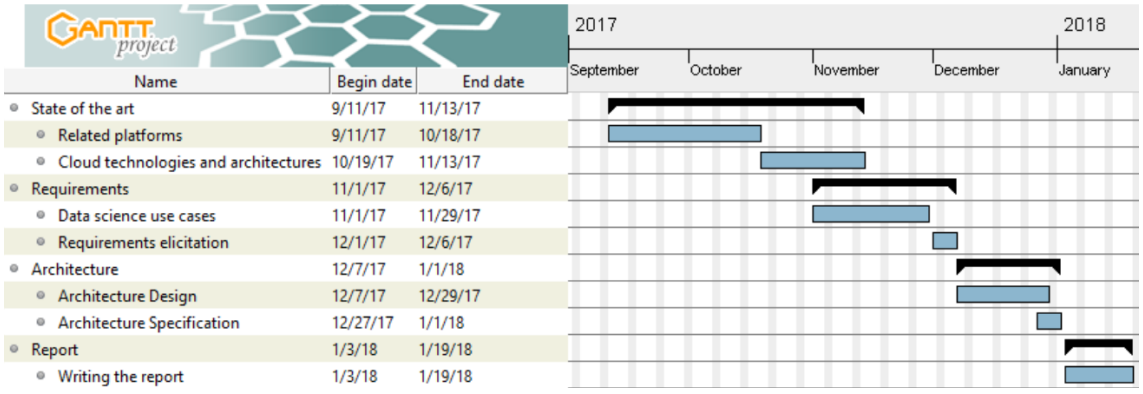

Figure 7.1: Gantt chart for the 1st semester

Figure 7.1 illustrates our work schedule. It was specified in the beginning of the semester and was accomplished, for the most part, on schedule. The writing of the intermediate report suffered a setback and was a little bit rushed at the end.

Before starting the second semester, several goals were scheduled as depicted in Figure 7.2

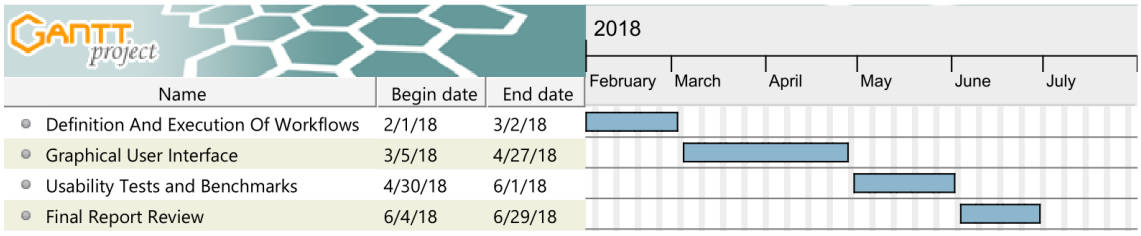

Figure 7.2: Gantt chart planned for the 2st semester

The first goal was to define a few example workflows and their execution without any

Grafical User Interface (GUI). This work would serve to prove that the execution could be done using **Condutor**. During this phase, **Condu** was developed as the necessary tool for the definition and execution of the workflows.

The next goal was to create a **GUI** that would allow the user to build and execute the workflows. With the exception of the 'data science services', every other microservice was implemented individually as the functionalities that we added to the GUI would require.

We saved May and June to do the usability and benchmark tests, and to write the final remaining documentation needed for the thesis. However, when May came we were still finishing the remaining requirements. We also got the opportunity to write two papers for two different conferences, which was a good opportunity to bring attention to this project. Because of this, the whole process was setback, causing the delivery of the report to be moved from July to September.

Figure 7.3 shows the setbacks suffered in relation to our planning.

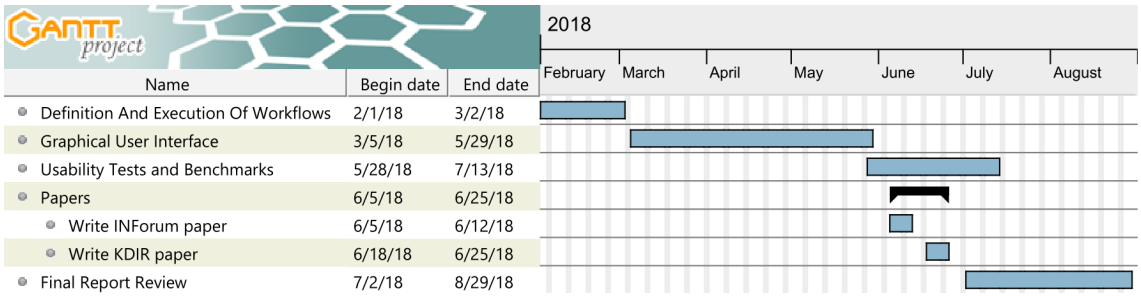

Figure 7.3: Gantt chart realised in the 2st semester

This page is intentionally left blank.

Chapter 8

Conclusion and future work

In this dissertation we presented a service for non-programmers to build data science experiments employing good data mining practices. We prototyped a cloud application that follows a microservices architecture. The use of an architecture that follows this pattern proved to be of great value to satisfy the quality attributes that were identified. The orchestration problem was one of the most interesting challenges that we had to overcome. It had a key role for the execution of the data mining processes and the distribution of that workload.

The interface built tried to achieve a high degree of usability. To test it, experiments were made with experienced and non-experienced users to evaluate the prototype and validate the paradigm of visual programming using sequential tasks.

The results were satisfactory with a positive feedback and without critiques related to what we aim to achieve. In the future we plan to add predefined data mining workflow templates that might be searched, changed and shared by the users, as well as make comparative benchmarks with more platforms and provide support for Big Data algorithms. Regarding the usability tests we plan to improve the application by making changes to the user interface according to the feedback received.

Future works will include not only more usability tests with experienced users to improve the user interface in aesthetics and functionality terms, but mainly the investment in optimising the current architecture, which will include improvements to the orchestration of microservices and exploring different solutions for the storage of datasets. These improvements will increase the performance and help minimise storage costs.

We were able to implement all the 'must' use cases. However, some use cases with lower priority and related with adding more Machine Learning (ML) algorithms were not possible to implement at this time and will be implemented in the future. These use cases are 1.4 and 1.7 - 1.10.

This page is intentionally left blank.

References

[1] ISO/IEC 9126-4. Software engineering - Product quality - Part 4: Quality in use metrics. Standard, International Organization for Standardization, April 2004.

[2] Edgar Anderson. The species problem in iris. *Annals of the Missouri Botanical Garden*, 23(3):457, sep 1936.

[3] Apache. Hadoop. `http://hadoop.apache.org/`. Accessed: July 18, 2018.

[4] Jaime Correia Filipe Araujo Jorge Cardoso Rui Pedro Paiva Artur Pedroso, Bruno Leonel Lopes. A data mining service for non-programmers. *KDIR 2018*, 07 2018.

[5] Viren Baraiya. Netflix conductor overview. `https://medium.com/netflix-techblog/netflix-conductor-a-microservices-orchestrator-2e8d4771bf40`. Accessed: October 20, 2017.

[6] Mario Barbacci, Mark H. Klein, Thomas A. Longstaff, and Charles B. Weinstock. Quality attributes. , 1995.

[7] Len Bass, Paul Clements, and Rick Kazman. *Software Architecture in Practice (3rd Edition) (SEI Series in Software Engineering)*. Addison-Wesley Professional, 2012.

[8] Zoë Bernard. Netflix's subscription service is growing by leaps and bounds. `https://www.businessinsider.com/netflix-subscription-subscribers-growing-charts-2018-4`, April 2018.

[9] Simon Brown. The c4 model for software architecture. `https://c4model.com/`. Accessed: May 24, 2018.

[10] Jaime Correia Filipe Araujo Jorge Cardoso Rui Pedro Paiva Bruno Leonel Lopes, Artur Pedroso. A data science service for non-programmers. *INForum 2018*, 06 2018.

[11] Longbing Cao. Data science: A comprehensive overview. *ACM Comput. Surv.*, 50(3):43:1–43:42, June 2017.

[12] Gavin C. Cawley and Nicola L.C. Talbot. On over-fitting in model selection and subsequent selection bias in performance evaluation. *J. Mach. Learn. Res.*, 11:2079–2107, August 2010.

[13] Microsoft Cesar de la torre. Microservices and docker containers architecture patterns and development guidance. `https://blogs.msdn.microsoft.com/dotnet/2017/08/02/microservices-and-docker-containers-architecture-patterns-and-development-guidance/`. August 2, 2017.

[14] Patrick Chanezon. Docker leads oci release. https://blog.docker.com/2017/07/oci-release-of-v1-0-runtime-and-image-format-specifications/. Accessed: January 15, 2018.

[15] Alistair Cockburn. *Writing Effective Use Cases*. Addison-Wesley Professional, 2000.

[16] Containous. Træfik. https://docs.traefik.io/. Accessed: June 20, 2018.

[17] Inc Docker. Docker documentation. https://docs.docker.com. Accessed: December 8, 2017.

[18] Thomas Erl. *Service-Oriented Architecture: Analysis and Design for Services and Microservices (2nd Edition) (The Prentice Hall Service Technology Series from Thomas Erl)*. Prentice Hall, 2016.

[19] Open Group. Service-oriented architecture – soa features and benefits. http://www.opengroup.org/soa/source-book/soa/p4.htm. Accessed: May 24, 2018.

[20] H2O. One of the nodes in my cluster is unavailable — what do i do? http://docs.h2o.ai/h2o/latest-stable/h2o-docs/faq/clusters.html. Accessed: August 10, 2018.

[21] Aaron L. F. Han, Derek F. Wong, and Lidia S. Chao. Password cracking and countermeasures in computer security: A survey. *CoRR*, abs/1411.7803, 2014.

[22] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York Inc., New York, NY, USA, 2001.

[23] T. Haynes and D. Noveck. Network file system (nfs) version 4 protocol. RFC 7530, RFC Editor, March 2015.

[24] Leonard Heiler. Difference of data science, machine learning and data mining. https://www.datasciencecentral.com/profiles/blogs/difference-of-data-science-machine-learning-and-data-mining. Published: March 20, 2017.

[25] Michael Chui James Manyika Tamim Saleh Bill Wiesman Henke Nicolaus, Jacques Bughin and Guru Sethupathy. Mckinsey global institute. In *The next frontier for inovation, and productivity*, December, 2016.

[26] Nikolas Herbst, Samuel Kounev, and Ralf Reussner. *Elasticity in Cloud Computing: What it is, and What it is Not*. Karlsruhe Institute of Technology, 06 2013.

[27] M. Jones, J. Bradley, and N. Sakimura. Json web token (jwt). RFC 7519, RFC Editor, May 2015. http://www.rfc-editor.org/rfc/rfc7519.txt.

[28] Washington Faculty Josh Tenenberg. Casual use cases. http://faculty.washington.edu/jtenenbg/courses/360/w11/docs/CasualUseCases.pdf. Published: January 6, 2011.

[29] Rick Kazman, Mark Klein, and Paul Clements. Atam: Method for architecture evaluation. Technical Report CMU/SEI-2000-TR-004, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 2000.

[30] Janez Kranjc, Roman Orač, Vid Podpečan, Nada Lavrač, and Marko Robnik-Šikonja. Clowdflows: Online workflows for distributed big data mining. *Future Generation Computer Systems*, 68:38 – 58, 2017.

[31] Damjan Krstajic, Ljubomir J. Buturovic, David E. Leahy, and Simon Thomas. Cross-validation pitfalls when selecting and assessing regression and classification models. *Journal of Cheminformatics*, 6(1):10, Mar 2014.

[32] Pavel Kukhnavets. Moscow method: the most successful prioritization technique for any project.
`https://blog.ganttpro.com/en/prioritization-techniques-and-methods-for-projects-with-advantages-of-moscow-model/`. Published: October 26, 2016.

[33] Viktor Medvedev, Olga Kurasova, Jolita Bernataviciene, Povilas Treigys, Virginijus Marcinkevičius, and Gintautas Dzemyda. A new web-based solution for modelling data mining processes. *Simulation Modelling Practice and Theory*, 03 2017.

[34] Steven Miller and Debbie Hughes. The quant crunch: How the demand for data science skills is disrupting the job market. *Burning Glass Technologies*, 2017.

[35] NASA. Software design for maintainability.
`https://oce.jpl.nasa.gov/practices/dfe6.pdf`. Accessed: Jun 26, 2018.

[36] NGINX. Web server.
`https://www.nginx.com/`. Accessed: June 24, 2018.

[37] Pivotal. Rabbitmq - message broker.
`https://www.rabbitmq.com/`. Accessed: February 6, 2018.

[38] Daniel Pop. Machine learning and cloud computing: Survey of distributed and saas solutions. *CoRR*, abs/1603.08767, 2016.

[39] Daniel Pop and Gabriel Iuhasz. Overview of machine learning tools and libraries, 12 2011.

[40] James Scott, Boeing Corporation, Rick Kazman, and Software Engineering Institute. *Realizing and Refining Architectural Tactics: Availability*. Software Engineering Institute, Carnegie Mellon University, August 2009.

[41] Weka. Mining big data with weka 3.
`https://www.cs.waikato.ac.nz/ml/weka/bigdata.html`. Accessed: August 5, 2018.